

CT³ High Level Specification

I. INTRODUCTION

This document is an attempt to provide a detailed specification of game play, client and server responsibilities, and configurable parameters of the Colored Trails framework. Anything not explicitly detailed in this specification should be assumed to be not included. Ambiguities in the current specification are footnoted, and probable answers to these ambiguities sometimes accompany the footnotes in italics. This specification is meant to be a high level, rather than implementation level, specification.

II. BASIC GAME PLAY

A. Number of Players

Colored Trails is played by two or more human or non-human players. There is no upper bound on the number of players.

B. Board

Colored Trails is played on a rectangular board.¹ The board consists of an $n \times m$ matrix of squares. Each square is assigned a solid color from a given palette of colors (e.g., red, blue, green). Two different types of objects may also be placed (non-exclusively) on top of squares—these are player *tokens* and *goals*.² A token represents one player's location on the board. Players cannot select how their token will be displayed (e.g., a star versus a longhorn versus a frog)—they are assigned randomly.³ Goals mark squares as an objective

¹ Are Colored Trails boards always square as opposed to rectangular? (*Probable: No.*)

² Can goals be specific to given players? Should CT³ have the capacity to have per-player goals? (*Probable: Yes, but how to display this?*)

³ Should tokens simply be numbers or something non-emotive? Would this in turn cause a *different* framing problem? (*Probable: No, should continue to use same tokens as previously.*)

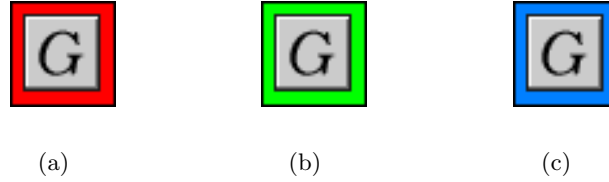


Fig. 1. Three goals with three different background colors determining the chip type that must be relinquished to reach them.

to be reached, rather than counting as a particular color of the palette (in other words, a goal square is marked with an icon of some sort, but it also has an original base color—see Figure 1). Multiple player tokens may be on the same square at the same time.

C. Chips

Each player is assigned a given set of *chips*. Chips are designated colors from the same palette as the board. Chips may be traded to another player, given to another player, or relinquished to permit movement to a square of the same color as the chip (e.g., a player relinquishes one green chip to move to a green square adjacent to the player’s current position).

D. Phases

Game play proceeds in a series of *phases*. Each phase has a *duration*, a *following phase*, and a set of *allowed actions*. The set of allowed actions can be specified on a per-player basis, so that one can specify that a given player (or set of players) can perform a given action in a given phase. A phase with no specified following phase ends the game on completion. Loops based on following phases are allowed, but not conditional loops (e.g., only one main loop is allowed) and all phases must be contained in the loop (see Figure 2).

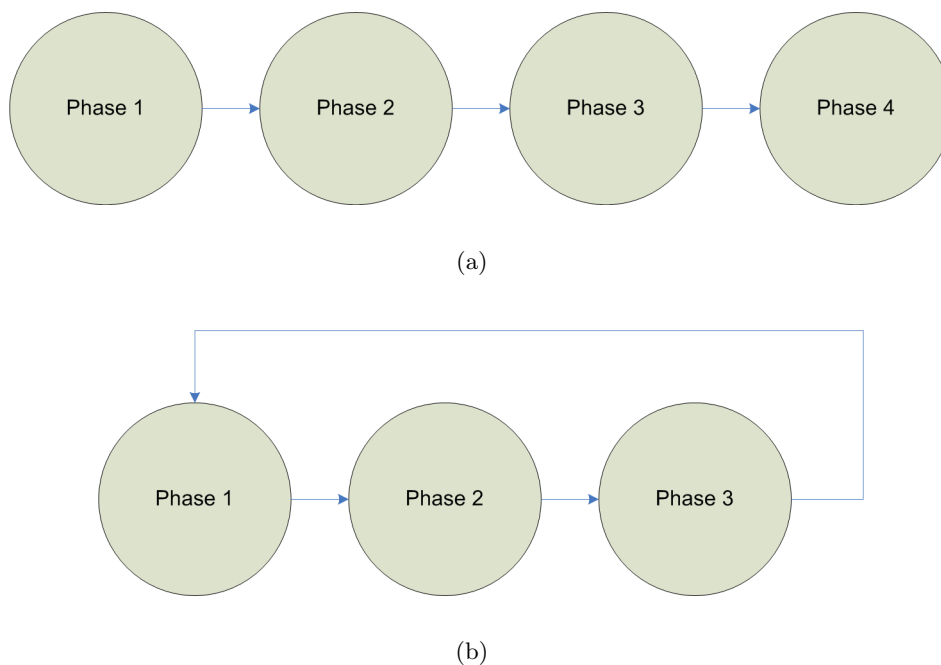


Fig. 2. There are two different types of phase sequences—non-looping, as in Figure 2(a), and looping (with the loop running over the entire sequence of phases), as in Figure 2(b). Note: The diagrams above are purely illustrative, and are not meant to be shown to the user.

E. Actions

There are four types of actions:

Propose exchange Offer a set of chips to another player in exchange for a different set of chips.

Accept exchange Commit to a proposed exchange.

Transfer chips Subtract a given set of chips from the player's set of chips, and add them to a specified other player's set of chips.

Move If legal, move to a given square by relinquishing a chip of the new square's color.

Only one move to a new adjacent square is permitted per phase.⁴

⁴ Should this be the case?

Exchange proposals and acceptances are sent to the affected players immediately. Chip transfers and player moves happen at the end of the current phase.⁵

F. Game Play

Game play consists of players executing allowable actions as defined by the present phase.

G. End of Game

The game ends when one of the following occurs:

1. A preset number of phases have passed (e.g., 10).
2. A phase has ended which has no following phase (e.g., the end of Phase 4 in Figure 2(a)).
3. There has been no movement by any player for a preset number of phases (e.g., no one has moved for 16 phases).⁶ This case applies specifically to movement and not to any action in general, otherwise computer agents making unsatisfactory proposals could lead to a game played forever.
4. All players have either reached the goal or left the game.

In the first and third cases, the preset number of phases is determined in advance as part of the initial description of the game (like the board colors or goal locations), see Section IV.I for more information.

⁵ Is this something that should be all or nothing, in other words, all actions occur immediately or at the end of the phase? Is the split between communication timing and action timing (i.e., with communication happening immediately and actions happening at the end of the turn) the proper split? It likely does not make sense to have dynamic (e.g., per game) configurability for when individual granular action types occur due to the complexity of implementation.

⁶ This is a departure from some of the currently often used rules (specified in the server input file), which drop a player after 3 movement phases of that player's inactivity. This behavior likely leads to a more complicated model of proper game play for an agent, since not getting the right chips quickly could lead to being dropped from the game.

H. Objective

The objective of the game is for a player to get the highest possible score.⁷ A player's score is a weighted sum (with weights preset at the beginning of the game) of:

1. Whether the player has reached the goal (i.e., 0 or 1 times the weight).
2. The final Manhattan distance of the player from the goal.
3. The number of chips the player has left.
4. The sum, average, minimum, and maximum of scores on the player's team.
5. The sum, average, minimum, and maximum of scores for all players.⁸

III. FURTHER GAME PLAY CONSIDERATIONS

A. Visibility of Chips and Board Squares

A player's own chips are always known. A game may be preset to allow or not to allow players to see other players' chips during game play.⁹ Board squares are always visible.¹⁰

B. Withdrawal

Players may withdraw from the game at any time, and their final score is calculated at the end of the game given their board position and ending chips (i.e., not at the time when they leave the game, but after all players have finished).

⁷ Does a player win a tournament by having the highest total score? Does a player win a tournament by having the most wins? In other words, is a player trying to maximize payoff in general, or are they trying to get a higher payoff than their competitors in a particular game? (*Probable: Maximize payoff in general—Gal's experiments paid participants linearly based on their total points at the end of several games, rather than on the number of games "won."*) This is as much a question for experimenters as it is a part of the game specification—experimenters may change the objective as they like, but I wanted to establish a general objective for most games here.

⁸ The last two (eight) values are ambiguous in several ways—at the very least, "score" for the purpose of calculating the group score must be made up only of non-aggregate (team, global) values, to avoid a circular dependence.

⁹ Should team members, and players more generally, be able to discuss what chips they have? What is the simplest set of primitives that would enable this?

¹⁰ It is still an open question how to handle board visibility with respect to teams, communication, the path finder display, and other features.

C. Forced Actions

A particular game is defined at the outset to be either a *non-compulsory exchange* or a *compulsory exchange* game. In the former, players may make and accept proposals without regard to what chips they actually have available, and may transfer whatever chips they see fit. In the latter, the set of the player's outstanding chips in offers or proposals committed to may not exceed the player's currently controlled chips. In other words, if a player has four green chips, he may have an offer of two green chips to one player, and an agreement with another player in which he owes another two green chips, but he may not make another offer of green chips, or accept another offer which requires sending green chips. When an offer is retracted, whatever chips were "tied up" in the exchange are now available again to the players for future offers. Any offer may be retracted at no cost.¹¹ In a compulsory exchange game, there is an "auto-exchange" phase. During this phase, all exchanges which have been agreed to are executed automatically by the server.¹²

The other type of forced action phase is an "auto-movement" phase. During an "auto-movement" phase, all players move as close to the goal as possible, given the chips they each have available.

¹¹ Should retractions be the default? Should no retractions be the default? Should retractions not be allowed after an agreement?

¹² This is a departure both in language and in action from the complex commitment protocol described in Nesson and Rayner. That protocol allows for a space of over 500 different types of commitments, obligations, and punishments for making conflicting commitments or not fulfilling a commitment. It also differentiates between the word "binding," which is meant to describe any game which has an auto-exchange phase, and the words "commitment" or "committed," where a commitment is an obligation based on having agreed through some protocol to send the chips. The commitment protocol described includes four variations on when agents are bound to their commitments, five variations on when an exchange becomes binding (confusing because of the equivocation with the word "binding"), four variations on how the system may handle conflicting commitments in terms of notification of other agents, four variations for how the system may automatically resolve conflicting commitments, and five different types of decommitment fees (fees in points or chips for not executing an offer). Such commitment protocols are an interesting area of future research, but for the present specification (and based on Gal's and Kraus' experiments) we will limit the exchange types to these two, with different names in order to avoid confusion.

D. Teams

A player may be grouped with one or more other players on a *team*. A player may only be on one team (e.g., teams are exclusive), though they may not be on a team at all, even in a game involving teams. Teams only determine final scoring, and do not affect game play. Teams are preset by the game configuration at the beginning of a game.

IV. SERVER

A. Administrator Interface

An administrator client can connect to the server and send a set of commands which are used to control whom is paired with whom to play games and which configurations are used. The administrator interface allows adding game configurations, listing current game configurations, listing current in progress and past games, listing currently connected players, and matching players to game configurations. An example of what a command line administrator client might look like is as follows:¹³

```
> add configuration (cfg2) (file2.cfg)
configuration added
> list configurations
cfg2 - 2 players, imported from file2.cfg
> list connected players
11112
11113
> new game (game1) config (cfg2) players (11112 11113)
```

¹³ This is not the exact syntax which will end up being used—that will be a question for the implementation specification. However, at least the functionality shown here will be available—adding configurations, creating new games, listing players, and listing games.

```

game started.

> list games

game1 - config=cfg2, players=(11112 11113)

```

This client interface may eventually allow on-the-fly modification of game configurations, though for the moment it will require a separate configuration file for each game configuration. No authentication of the administrator interface will occur—it is presumed that CT³ is an experimental framework rather than a game framework, and as such that players will not actively be trying to cheat. As a result, validation will still need to be performed by the server to prevent broken clients from making impossible moves and to disallow trivial cheating by the clients, but no attempt will be made (at least initially) to protect games from being modified by clients connected to the administrator interface.¹⁴

B. Per Game Information

The server needs to keep track of all board, chip, and player location information for each game. This is necessary in order to validate actions by players. It will also keep a complete log of actions which have occurred during the game. Games are identified by the server using a *gameid* and clients communicate to the server about a particular game they are playing using this *gameid*.

C. Multiple Games

Multiple games are enabled by assigning each game a *gameid*, and by the manual connection of players to games using the administrator interface (though automatic clients for this interface could be written later).

¹⁴ It may simply work not to allow administrator commands from anywhere but the local server, but as noted, protecting the server from abuse is not an interesting question in the vein of research which CT³ is designed for.

D. Relaying Messages

The server is in charge of relaying communication messages between clients. These messages can include any data desired, and the server will simply pass on messages unchanged except for the two types of messages it understands:

1. Chip exchange proposals.
2. Chip exchange acceptances/replies.

This is necessary in order to ensure that exchanges can be made compulsory and that players may not agree to or offer conflicting exchanges in the compulsory case. Communication happens using the *discourse* primitive discussed in Section V.B.

E. Information Requests

At any time, a client may request certain data—players in the game, board configuration, and so forth. This data is filtered based on what the client should see given the game configuration (i.e., the server should not send chip data for other players when this data is meant to be invisible). Information requests are discussed in Section V.A.

F. Actions

Clients can send action requests to the server at any time during the phase. These requests will be validated by the server to make sure the action is legal. Legality is determined as:

1. A movement action does not require relinquishing a chip which the player does not own.
2. The proposal or acceptance of an exchange does not result in the set of chips allocated for offers and acceptances being greater than the player's available chips.
3. A chip transfer does not involve transferring chips not controlled by the player trans-

ferring.

4. The player is permitted to perform the necessary action given his or her role in the present phase.

If legal, actions are placed on one of two queues—actions to happen at the end of the current phase, and actions to happen immediately. Action requests are discussed further in Section V.B.

G. Updates

The server will process the actions in the immediate queue immediately and the end of phase queue at the end of the current phase. Processing involves updating the game state to reflect changes due to the action, as well as sending a message to each client describing the update which took place (or sending a message noting that an update happened, such that the client can request an update explicitly).

H. End of Game

The server will either periodically or at the end of each phase determine if one of the conditions which entails the end of the game has occurred. An update is sent to clients announcing this fact.

I. Game Configuration Format

The game configuration format will allow for the following points of configuration:¹⁵

Board The size of the board, initial locations for each player, goals, and colors for each square.

¹⁵ Although the following are the only planned points of configurability, the configuration format should be designed to allow miscellaneous options (either as key/value pairs or as more complicated structures) and ideally would support some sort of game specific code, though the latter would be significantly more difficult. This game specific code might more easily allow actions with add-on side effects, like proposals that end the current turn.

Players The number of players involved in the game, what their allowed actions are (roles), and their starting chips.

Phases The phases that will make up the game (as described above), and what actions are allowed by which players in each phase.

Scoring Coefficients for determining players' scores at the end of the game.

End of Game Whether a specific number of phases or move-less phases ends the game.

Visibility Whether other players' chips are visible (this is all-or-nothing in that either all players' chips are visible to all other players, or they are visible to no one but the players who control them).

Teams The team designation of each player, if any.

Exchange Type Compulsory or non-compulsory exchanges.

Anything not described to be configurable should be assumed not to be.¹⁶ The game configuration format is read in from the administrative interface described above.

V. COMMUNICATION LANGUAGE

A. Information Primitives

The game will have the following game information primitives that the server can send the client:

New game changed A new game has begun.

Chips changed A change in the chip set of a player has occurred.

Board changed The board has changed, either because someone moved, they withdrew, or they timed out.

Phase changed The current phase has passed.

¹⁶ As noted above, it may make sense to allow configurability of whether messages are relayed immediately or at the end of a phase for a given game, but not at the granularity of individual messages. It may also make sense to describe a subset of allowed (understood) communication primitives for the game—one game might include discussion of reputation, whereas another might involve more planning primitives.

End game changed The current game has ended.

The client in turn can get information with the following primitives:

Get player IDs Get the IDs of all players currently in the game.

Get board Get the color of each square on the board and its dimensions.

Get player location Get the location of a player on the board.

Get player chips Get the chip set that the player currently has. Rejected by the server if the chips are not visible and the request is about another player.

Get player info Get miscellaneous information about a player, including their status in the game (withdrew, still playing, timed out, etc.).

Get phases Get the full list of phases in the game, and the current phase.

Get score calculation/coefficients Get the coefficients necessary to compute the client's present and future scores.

Get game parameters Get a list of miscellaneous game parameters with simple formats, like visibility or exchange type.

Get action history Get a list of actions and when they occurred.

Each of these messages is processed immediately.

B. Action Primitives

The client can send the following actions to the server:

Move Request that the server move the player's token to a specified adjacent square.

Transfer chips Request that the server transfer specified chips from the player to a designated player.

Withdraw from game Leave the game. The player's token still remains but the player has disconnected.

Discourse with one or more agents Request that the server send a communication mes-

sage to another player or players whenever it is next allowed.

Register for game Request that the server assign the player to a game.

The second to last action, *discourse*, circumscribes all agent-to-agent communication.

C. Administrator Interface Primitives

The administrator client can send the following actions to the server:

Add Configuration Add a game configuration to the server, and get its assigned ID.

List Configurations Get a list of current configuration IDs.

Get Configuration by ID Get the full information about a configuration by querying by ID (equivalent to the initial configuration file).

List Connected Players Get a list of connected players, their status, permanent ID, and possibly further information (like IP address).

List Games Get a list of game IDs and the status of each.

New Game Start a new game with a particular set of players and a particular game configuration.

Each of these messages is processed immediately by the server.

VI. CLIENTS

A. Specification of Connection Data

Clients allow the configuration of the server, server port, and the player ID of the connecting player.

B. Library of Algorithms

Clients will be able to take advantage of a library of algorithms to handle common tasks in writing a CT³ agent. This library will include:

1. Shortest paths calculation, given a board and an initial location.

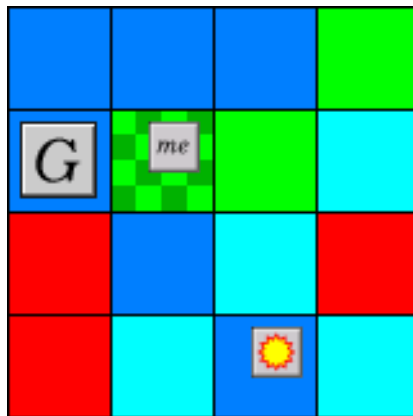


Fig. 3. The board shows squares, player tokens, and goals. At present, the player's current square is emphasized by a partial fill. Movement is currently handled by dragging, with available squares for moving emphasized with a partial fill during the drag.

2. A*, given a board, chips, and an initial location.

The library may also include more general code helpful in getting a client running quickly.

In other words, ideally the human client should be as non-specific to humans as possible, such that other agents can reuse most of the generic code necessary to build that client.

C. Human Client

The human client will consist of several features, as described below.

C.1 Board Display

Human players require a view of the game board. This is accomplished using a game board display (Figure 3) which includes the colors of the squares as well as overlayed game and player token sprites. The board display also allows the player to move during a movement phase—the previous version worked via drag and drop, with players dragging their token into an adjacent square which they are allowed to move to, and the version prior used radio buttons to permit adjacent movement. This specification does not define how movement should occur in the future, but either of the above methods is sufficient.

Player				
<i>me</i>	0	1	3	1

Fig. 4. This diagram shows the current player chip display. CT³ does not necessarily need to use this same display, but this one is adequate. Numbers delineate the number of each chip that the player controls.

C.2 Chips Display

One piece of information that the player must know at all times is the collection of chips that he or she currently possesses. This is accomplished using the same chip set panel which is used for other chip sets (Figure 4).

C.3 Path Finder

The path finder allows the user to graphically see the shortest paths to the goal, and the chips which the user is missing in order to reach the goal. Shortest is defined as requiring the least of a particular group of chips (e.g., two green chips is not less than nine purple chips, so both would be considered shortest).¹⁷ This specification suggests that the current path finder (Figure 5) continue to be the standard one.

However, the path finder should also permit a view of how far along a particular path the user can go at present. If two paths are missing the same amount of chips to reach the goal, but one has the chip missing at the end of the path, and the other has the chip missing at the beginning, then the user should see this graphically. This requires that the lines drawn on the board by the current path finder be modified to only show as far as they can reach with the current chips. A dotted line (Figure 6) should show the rest of the route, if the player was to have the remaining chips.

While the path finder could easily be extended to give the player more information about

¹⁷ It is unclear (a) whether this frames the problem in a particular way for players and (b) whether it makes more sense to give an optimal score based on trades interface of some sort.

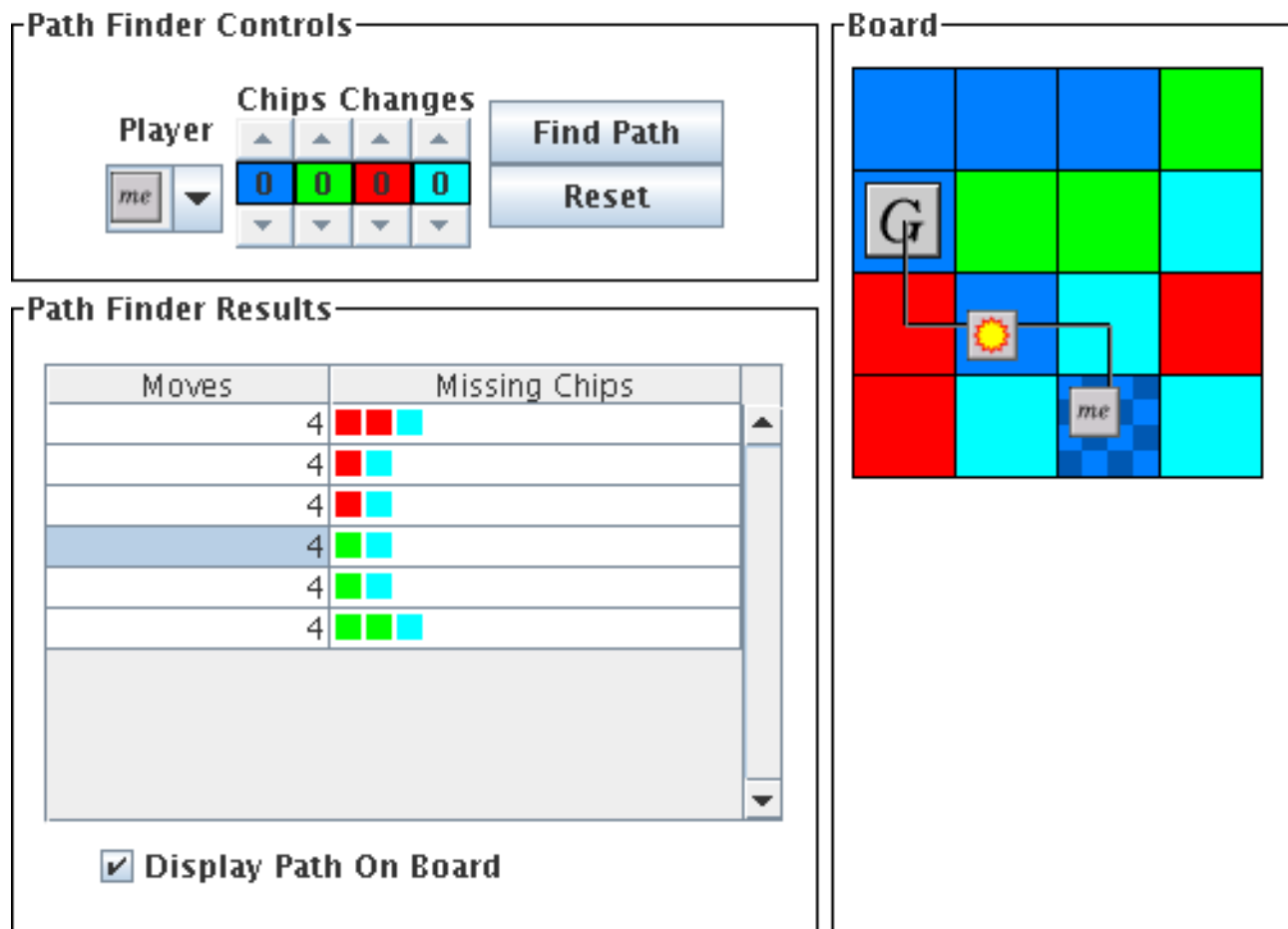


Fig. 5. This diagram shows the current CT2 path finder. The player can add or remove chips to see what potential trades might allow. The path finder lists all optimal paths (as defined by shortest paths above), and the number of chips missing for the path. If a particular path is clicked, and the “display on board” option is checked, the path to the goal is displayed directly on the board.

maximizing their score or determining optimal results from particular available trades, it should not be extended in this manner. The goal of the path finder is to provide the player with more information to make qualitative, rather than quantitative, decisions.

C.4 Phase Information Display

The player needs to know which phase is currently in progress, what the duration of the phase is, and what phases are coming in the future. CT³ will use a phase information

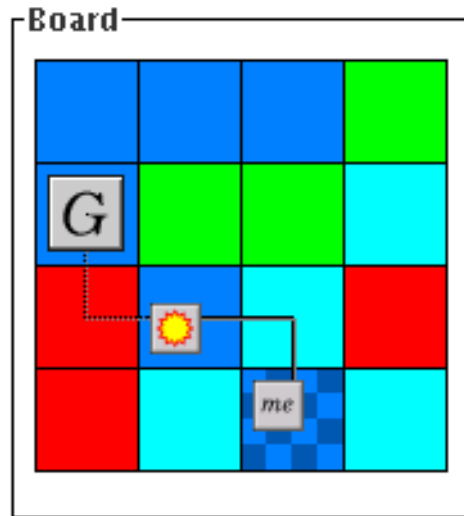


Fig. 6. This diagram shows the board segment of the path finder when the player lacks a red chip, so that the player could only execute the first two moves in the path. A solid line delineates the available length of the path, and a dotted line delineates the remaining segment of the path leading to the goal.

dialog (Figure 7) to convey this information. The current phase is either shown in bold or in regular type, with its duration ticking down, while other phases are shown in grey, with their total duration displayed.

D. Taskbar

Informing the user of presently available actions is important in human experiments. Previously, CT2 was modified in such experiments to pop-up proposal and acceptance windows automatically. This was not easily configurable, and was somewhat obtrusive. Alternatively, CT2 could just be left in its default state, which greyed out unavailable menu options, but required the user to scan through the menus to find which actions were available. Instead, CT³ will use a graphical taskbar (Figure 8), featuring proposal, acceptance, and transfer options, to ease such experiments by providing the user with a clear but unobtrusive view of currently available actions.

Phase Information	
Phases	Time Left
Movement Phase	3:00
Communication Phase	2:12
Exchange Phase 1	3:00
Exchange Phase 2	3:00
Movement Phase	3:00

(a) Looping Phase Information

Phase Information	
Phases	Time Left
Movement Phase	3:00
Communication Phase	2:12
Exchange Phase 1	3:00
Auto Exchange	AUTO

(b) Finite Phase Information

Fig. 7. The phase information display displays information about present and future phases to the user. Figure 7(a) shows what a normal, looping phase display looks like. When a phase passes, all phases shift up one cell, and a new phases appears at the bottom. In this way, the display shows the previous and next three phases. Figure 7(b) shows what a finite phase display looks like when the end of the game is coming up—no phases appear after the last phase. It also shows how automatic phases (auto-exchange and auto-movement) will appear—they are phases, but have no duration and have “AUTO” listed next to them.

D.1 Other Players Display

This display will show relevant information about other players. This includes chip sets, allowed actions in the current phase (and possibly future phases), and the token they are identified by. A mockup for this display is pending.¹⁸

¹⁸ It is currently unclear whether this display should look more like the horizontal message history display (Figure 10), with columns for player icon, player chips, and player role (with a team color in the background of the row) or

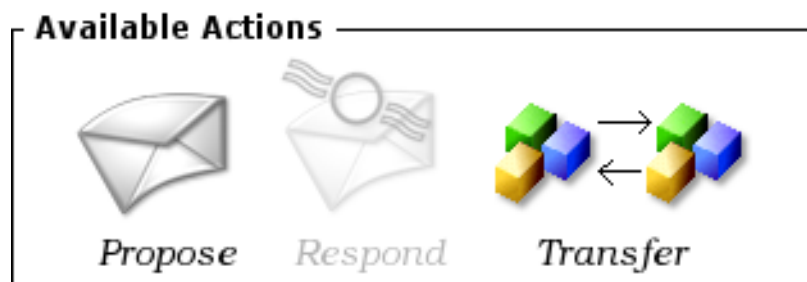


Fig. 8. A graphical taskbar showing the available actions during a given phase.

D.2 Exchange Display and Message History

The core of the CT³ game is chip proposals, exchanges, and transfers. Proposals and transfers are accomplished using dialogs (Figures 9(a) and 9(b)) which are mostly adequate, but should be extended in the future to allow messages to multiple players. On receipt, an exchange proposal or transfer is added to the message history window (Figure 10). The message history window both allows responses to messages (acceptances, rejections) and serves as a log of past offers, responses, and transfers.¹⁹

APPENDICES

A. UPDATES AFTER INITIAL DISCUSSION OF THIS SPECIFICATION

After a number of discussions of the initial specification (above), there were a number of changes and additions that various parties have asked for. These are as follows:

1. CT³ should implement the ability to have multiple, per-player goals and this should be implemented in an intelligent way (with respect to the path finder, scoring, etc.).

whether it should look more like a vertical buddy list (or like the previous message history pre-CT2), with each player having several lines, one for their icon and team, one for their responsibilities, and one for their chip display. The former might be easier in the short term, but the latter would permit additional amounts of arbitrary information about a player in an ad-hoc fashion (depending on the experiment).

¹⁹ What is the full range of acceptances and rejections that a player might send in a game of CT? *Discourse*, above, permits a full range of acceptances or rejections, but the human client will only be able to make a few of these acceptance and rejection types available before the user becomes confused—what should they be?

2. CT³ should allow for independent setting of enforced agreements and the ability to retract proposals or acceptances.
3. CT³'s server should send data when state is updated rather than just sending a notice that an update has occurred.
4. CT³ should allow for repeated games, i.e., allowing players to have identity that will be kept for a few games and a way for an agent to save information from one game to another.
5. If there is no visibility, the negotiation language should be extended to allow for discussion what chips one has.

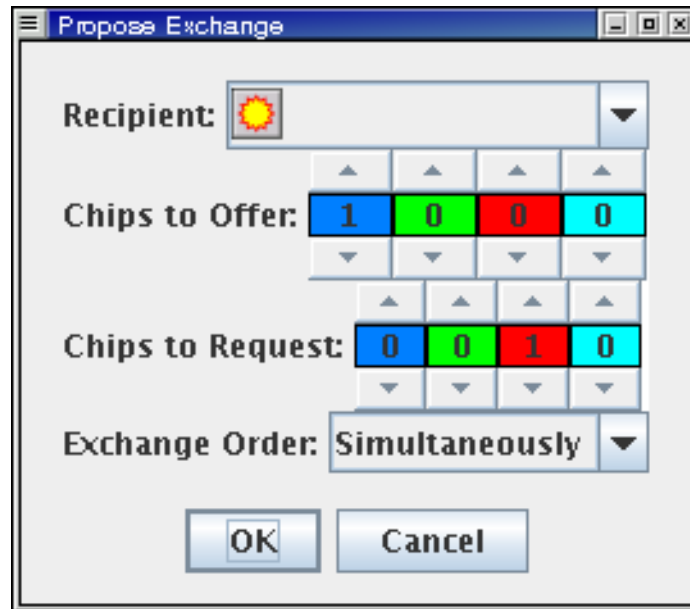
B. DEPRECATED FEATURES

The following features have been deprecated—they were previously part of the CT specification or code base, but are not featured in this one. They include:

1. Claim that another player is a liar.
2. Suggest an alternate path.

C. FUTURE QUESTIONS AND IDEAS

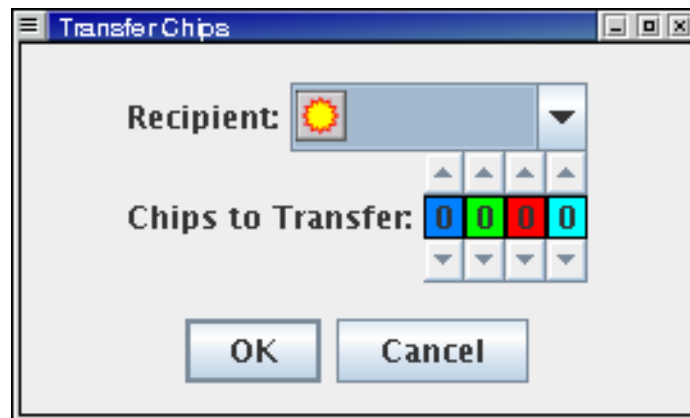
How should teams affect visibility and other features? Should visibility (of chips, the board, and so forth) be assumed, or should it be communicated with some amount of uncertainty? How should board visibility be implemented with respect to teams, the path finder, the HCI in general, and movement? How could teams be made dynamic rather than static?



The 'Propose Exchange' dialog box features a title bar with a menu icon, a close button, and a maximize button. The main area contains a 'Recipient' field with a sun icon and a dropdown arrow. Below this are two sets of four spinners each, labeled 'Chips to Offer' and 'Chips to Request'. The 'Chips to Offer' row shows values 1 (blue), 0 (green), 0 (red), and 0 (cyan). The 'Chips to Request' row shows values 0 (blue), 0 (green), 1 (red), and 0 (cyan). An 'Exchange Order' dropdown is set to 'Simultaneously'. At the bottom are 'OK' and 'Cancel' buttons.

Field	Value 1	Value 2	Value 3	Value 4
Chips to Offer	1	0	0	0
Chips to Request	0	0	1	0

(a) Make Proposal Dialog



The 'Transfer Chips' dialog box has a title bar with a menu icon, a close button, and a maximize button. It includes a 'Recipient' field with a sun icon and a dropdown arrow. Below it is a single row of four spinners labeled 'Chips to Transfer' with values 0 (blue), 0 (green), 0 (red), and 0 (cyan). 'OK' and 'Cancel' buttons are at the bottom.

Field	Value 1	Value 2	Value 3	Value 4
Chips to Transfer	0	0	0	0

(b) Transfer Dialog

Fig. 9. These are the current CT2 proposal and transfer dialogs. They could be better, but will be fine for the present. One feature which would be helpful is if they allowed making a proposal to multiple recipients easily.

Turn	Type	Sender	Recipient	Chips I ...	Chips I ...	Sends first	Action	Reply
9	Proposal		<i>me</i>			.	Do ...	
9	Proposal		<i>me</i>	 	 	.	Do ...	
10	Proposal	<i>me</i>		 	 	.	Do ...	
10	Proposal	<i>me</i>		 	 		Do ...	

Fig. 10. The message history shows both past actions and offers and offers pending responses. In the future, it would be ideal to reduce the number of responsibilities that the message history has—one example would be creating a separate proposal response display, such that the player can easily differentiate between previous actions and new proposals requiring a response, as well as more easily allowing many different types of rejections or acceptances.