BW4T2 Specification

March 21, 2012

This document describes the interface we designed for the Blocks World for Teams 2 (BW4T2) environment, allowing software agents to control robots that are part of this simulated environment.

The environment consists of nine rooms and a drop zone that are connected through halls (see Figure 1 and 2). Colored blocks are placed inside the rooms. The robot team should pick up blocks from the rooms, bring them to the drop zone and put them down there, in the color sequence as specified at the bottom of the interface. Blocks only become visible once a robot enters the room where blocks are. Robots cannot see each other. Once a robot enters a room (including the drop zone), no other robots can enter. Blocks disappear from the environment when dropped in the hall or in the drop zone.

Robots can be controlled by agents or humans (both are referred to as players), thereby providing the possibility to investigate human-agent robot teamwork. The focus of this document is on agent teamwork. Agents can communicate by sending messages to each other.

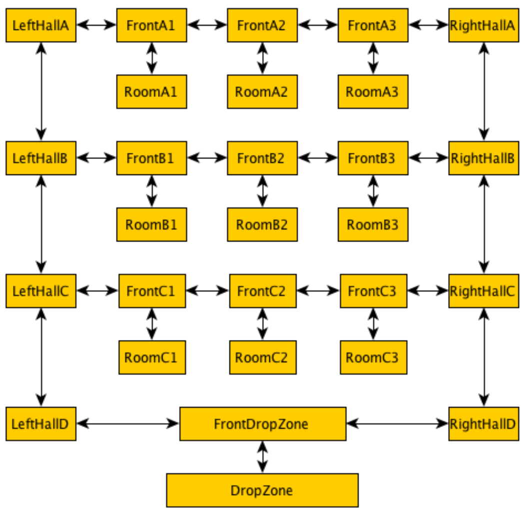

Figure 1. Rooms and connections (default Map1)

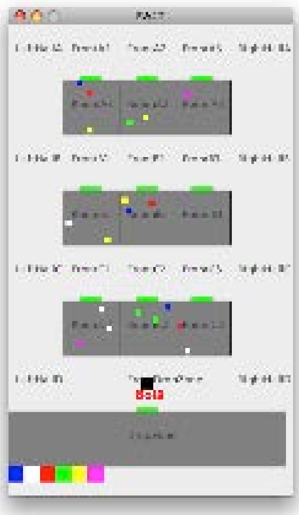

Figure 2. Overview window of the BW4T environment

Domain Labels

An ontology is needed for being able to represent the BW4T environment. In particular, we need labels, e.g., for blocks and rooms, as part of our language so players can talk about the environment. Below, we list these labels, and their parameter notations to be used in actions, percepts and messages.

Players

- Description: Players are the humans or agents that steer the robots in the BW4T environment.
- Parameter: `<PlayerID>`
- Values: Alphanumeric strings starting with a letter.

- Additional explanation: Each player is connected to one simulated robot in the environment and vice versa. Robots can drive around and pick up and drop blocks.

Blocks

- Description: Blocks should be picked up and dropped in the right order in the drop zone.
- Parameter: `<BlockID>`
- Values: Any natural number.
- Additional explanation: Block labels enable agents to distinguish between same-colored blocks.

Places

- Description: Places are rooms, hallways and the drop zone
- Parameter: `<PlaceID>`
- Values: All values of `<RoomID>` (see below) and `FrontDropZone, LeftHallD, LeftHallC, LeftHallB, LeftHallA, RightHallD, RightHallC, RightHallB, RightHallA, FrontA1, FrontA2, FrontA3, FrontB1, FrontB2, FrontB3, FrontC1, FrontC2, FrontC3.`
- Additional explanation: Figure 1 shows the different places and the way they are connected.

Rooms

- Description: Rooms are a subclass of places with special properties, i.e., a room can be occupied.
- Parameter: `<RoomID>`
- Values: `RoomA1`, `RoomA2`, `RoomA3`, `RoomB1`, `RoomB2`, `RoomB3`, `RoomC1`, `RoomC2`, `RoomC3`, `DropZone`.
- Additional explanation: Rooms and the drop zone allow only one agent to be in there at a time. For details on dropping blocks, see the action specification below.

Colors

- Description: Blocks have colors. Blocks should be delivered to the drop zone in an order specified by color, e.g., first deliver a blue block, then a red block, etc.
- Parameter: `<ColorID>`
- Values: `Blue, Cyan, Magenta, Orange, Red, White, Green, Yellow, Pink.`

State

- Description: A controlled robot can be in three different states, *traveling* if moving to a new location, *collided* if moving to a location failed and *arrived* if it arrived on its target location.
- Parameter: `<State>`
- Values: `collided, arrived, traveling`

Doors are not provided with names here, i.e., they cannot be referred to directly by players. Doors are associated with a *unique* room. Doors have special behavior and act as guards: all doors to a room "close" simultaneously when someone enters the room (from the outside perspective), but from the inside perspective in a sense there are no doors (effectively, you can always leave a room without having to open doors).

Percepts

The predicates introduced here can be used by agents to represent features of the BW4T environment. Instantiations of these predicates are provided as *percepts* to an agent (when it is connected to a robot in the environment).

Percept are separated in four different types:

1. Send once, these percepts will only be sent at the start of a simulation.
2. Send on change, these percepts will only be resent if there was a change.
3. Send on change with negation, for these percepts a not(percept) will be sent when an earlier percept no longer holds and vice versa.
4. Send always, these percepts will always be sent.

Predicate: `place(<PlaceID>)`

- Description: Information about which places there are in the environment.
- Translation: "<PlaceId> is a place."
- Type: Send once.
- Additional explanation: The interface does not provide the information as to which places are rooms.
- Example: `place('RoomA1')`.

Predicate: `in(<RoomID>)`

- Description: This player is in <`RoomID`>.
- Translation: "This player is in <`RoomID`>."
- Type: Send on change with negation.
- Example: `in('RoomC1')`
- Additional explanation: The percept is local, i.e., it is sent only to the player for which it holds. The percept is only sent when a player enters or leaves a room, i.e., the nine rooms and the drop zone, not for other places.

Predicate: `at(<PlaceID>)`

- Description: This player is at <PlaceID>.
- Translation: "This player is located at/near <PlaceID>."
- Type: Send on change.
- Additional explanation: Percept is local, i.e., is sent only to the player for which it holds.
- Example: `at('RightHallB')`

Predicate: `atBlock(<BlockID>)`

- Description: This player is at the location of block <BlockID>.
- Translation: "This player is located at block <BlockID>."
- Type: Send on change with negation.
- Additional explanation: Percept is local, i.e., is sent only to the player for which it holds. When robots are located at a block, it means they are close enough to pick it up.
- Example: `atBlock(3)`

Predicate: `occupied(<RoomId>)`

- Description: Holds if a robot is in a room with name <RoomId>.
- Translation: "Room <RoomId> is occupied."
- Type: Send on change with negation.
- Additional explanation: Percept is global, i.e., if a robot enters a room, the percept is sent to *all* players. Whenever a room is occupied it cannot be entered by anyone, i.e., from the perspective outside the room all doors of that room are closed.
- Example: `occupied('RoomA1')`means that you cannot enter room A1.

Predicate: `color(<BlockID>,<ColorID>)`

- Description: Information about the color of blocks.
- Translation: "Block <BlockID> has color <ColorID>."
- Type: Send on change with negation.
- Additional explanation: Percept is local, i.e., is sent only to the player who is in the same room as the blocks.
- Example: `color(1,red)`means that box 1 is red.

Predicate: `holding(<BlockID>)`

- Description: Information on the block this player is holding.
- Translation: "This player is holding block <BlockId>."
- Type: Send on change with negation.
- Additional explanation: Percept is local. It is sent only to the agent for which it holds. Robot can hold at most one block at a time.
- Example: `holding(1)`.

Predicate: `player(<PlayerID>)`

- Description: <PlayerID> is a player.
- Translation: “<PlayerID> is a player.”
- Type: Send once.
- Additional explanation: Only names of other players are sent. The player’s own name is sent with the `ownName(<PlayerID>)`percept. We talk about players because we prefer referring to the controlling entity instead of to the controlled entity, i.e. the robot. Each player is assigned exactly one robot to control.
- Example: `player('Bob')`.

Predicate: `ownName(<PlayerID>)`

- Description: <PlayerID> is this player's name.
- Translation: “<PlayerID> is this player's name.”
- Type: Send once.
- Additional explanation: We talk about players because we prefer referring to the controlling entity instead of to the controlled entity, i.e. the robot. Each player is assigned exactly one robot to control.
- Example: `ownName('Bob')`.

Predicate: `sequence([<ColorID>])`

- Description: Sequence of colors; this predicate is used to communicate the goal, i.e., the color order in which blocks have to be dropped at the drop zone.
- Translation: “Blocks should be dropped in the following order: `[<ColorID>]`.”
- Type: Send once.
- Additional explanation: At the start of the game, the entire sequence is sent. The percept is global, i.e., sent to all players.
- Example: `sequence([Blue,Red,Blue])`

Predicate: `sequenceIndex(<Integer>)`

- Description: Expresses the index of the currently needed color in the sequence. The first index of the sequence is 0.
- Translation: “Goal <Integer>-1 is met, on to goal <Integer>.”
- Type: Send on change.
- Additional explanation: The percept is global, i.e., sent to all players. The sequence index is also increased and sent when the last block of the goal sequence has been delivered. Players should thus infer themselves that the team goal has been achieved.
- Example: `sequenceIndex(3)`

Predicate: `state(<State>)`

- Description: The state that the controlled robot is in.
- Translation: “Robot is in state <State>”

- Type: Send on change.
- Additional explanation: Percept is local. It is sent only to the agent for which it holds. Possible states are *collided*, *arrived* and *traveling*.
- Example: `state(collided)`

The BW4T environment also sends percepts about the id's of blocks and robots, e.g. `block(<BlockID>)`and `bot(<BotID>)`. However, these percepts are usually not needed in your implementation and can be ignored.

Basic Actions

This section describes the basic actions that are available to an agent to control a robot in the BW4T environment.

Action: `goTo(<PlaceID>)`

Precondition: <PlaceID> must be a place.

Postcondition: You are located at the specified <PlaceID> if no obstacles to going to that location are present.

Additional explanation:

- The goTo action is successfully terminated when location <PlaceID> is reached, the percept state('Arrived') is sent.
- If doors are closed, a robot may not be able to go to a location, and in that case the goTo action fails and the robot will be close to the door where the action failed. A percept state('Collided') is sent when the goTo action fails this way.

Action: `goToBlock(<BlockID>)`

Precondition: <BlockID> must be a block.

Postcondition: You are located at the specified <BlockID> if no obstacles to going to that location are present.

Additional explanation:

- The goToBlock action is successfully terminated when location <BlockID> is reached, the percept state('Arrived') is sent.
- If doors are closed, a robot may not be able to go to a location, and in that case the goTo action fails and the robot will be close to the door where the action failed. A percept state('Collided') is sent when the goTo action fails this way.

Action: `pickUp`

Precondition: Robot is close to a block and does not hold a block yet.

Postcondition: Robot is holding the block, and the block is not located anywhere (i.e., there is no "at" percept for the block) until it is dropped.

Additional explanation: A robot always is closest to at most one block, and blocks are not stacked on top of each other. A percept `holding(<BlockID>)` will be provided to the agent if the action is successful.

Action: `putDown`

Precondition: Robot is holding a block at location <PlaceID>.

Postcondition: (i) If <PlaceID> is a room, the block is dropped within a tolerance range of the current position of the robot. (ii) If <PlaceID> is not in a room and not at the drop zone, then the block leaves the environment and (iii) if <PlaceID> is the drop zone, the block is also removed from the environment; in case it matches the current color needed according to the sequence predicate (representing the goal of the game) then this sequence (goal) is updated as well correspondingly.

Additional explanation: Action is simply ignored by environment when robot is not holding a block.