The Blocks World

May 16, 2008

Contents

Chapter 1

The blocks-world package

This package contains the source code of chapter 21, *"The Blocks World with Classes and Methods"* from Lisp (3rd edition) by Winston and Horn.

1.1 A picture of the world

The block objects represent a world that "looks" like this:

```
/----\    ^    /---------\      ^
| b4 |   /  \  |         |     / \
\____/  /_w7_\ |         |     / \
/----\  /----\ |         |    /   \  /--------\        /^\
| b1 |  | b2 | | b3      |    /   \  | b6     |       (18 )
\____/  \____/ \_________/   /_w5__\ \________/        \./
+-----------------------------------------------------------+
|                                                           |
+-----------------------------------------------------------+
```

1.2 Example

In the initial configuration, where all blocks have been placed directly on the table (not shown), `put-on` will move the objects like this:

```
BLOCKS-WORLD> (put-on b1 b2)
Move hand to pick up B1 at location (1 2).
Grasp B1.
Removing support relations between B1 and TABLE.
Move B1 to top of B2 at location (2 2).
Adding support relations between B1 and B2.
Ungrasp B1.
T
```

1.3 The different kinds of blocks

In this section:

- basic-block
- load-bearing-block
- movable-block

Movable blocks than can be moved onto load supporting blocks. Using multiple inheritance, there are also blocks that can do both.

`basic-block` [*Class*]

Superclasses

common-lisp:standard-object, sb-pcl::slot-object, common-lisp:t

Documented Subclasses

load-bearing-block , movable-block

Direct Slots

height —

name —

supported-by —

width —

position —

Details

The superclass of all objects in the Blocks World (not including the hand).

Subclasses of `basic-block` characterize different kinds of objects, and have different properties.

They all have a name, given as `block-name` and in the examples from the book, a global variable of that name is used to refer to them.

Since this chapter is an explanation of CLOS, no specific constructor function is defined, and users may call `make-instance` directly.

Slot Access Functions

- block-name

- block-width
- block-height
- block-position
- block-supported-by

`load-bearing-block` [*Class*]

Superclasses

basic-block , common-lisp:standard-object, sb-pcl::slot-object, common-lisp:t

Documented Subclasses

brick , table

Direct Slots

support-for —

Details

The superclass of objects in the Blocks World that other blocks can be placed onto.

This class is mixed into most blocks, except for the `wedge` and the `ball` .

Slot Access Functions

- block-support-for

Inherited Slot Access Functions

- block-name
- block-width
- block-height
- block-position
- block-supported-by

See also

- wedge
- ball

`movable-block` [*Class*]

Superclasses

basic-block , common-lisp:standard-object, sb-pcl::slot-object, common-lisp:t

Documented Subclasses

ball , brick , wedge

Direct Slots

None

Details

The superclass of objects in the Blocks World that can be moved by the hand. This class is mixed into all blocks except for the `table` .

Inherited Slot Access Functions

- block-name
- block-width
- block-height
- block-position
- block-supported-by

See also

- table

1.4 Block properties

In this section:

- block-name
- block-position
- block-width
- block-height
- block-supported-by
- block-support-for

Slot readers:

`block-name` *instance* [*Function*]

ARGUMENTS

instance — a `basic-block`

RETURN VALUES

a symbol

DETAILS

Returns the block's name, a symbol.

In the examples from the book, a global variable of this name is used to refer to `instance`.

SEE ALSO

- basic-block

`block-position` *instance* [*Function*]

ARGUMENTS

instance — a `basic-block`

Return Values

a list of two integers

Details

Returns the block's position.

The position of a block is specified as a list of its x and y coordinates, where the first axis runs along the table, and the second axis points upwards towards the hand.

Together with the block's width and height, the position determines which parts of the world this block occupies. No other objects can be placed to an overlapping position.

See also

- basic-block
- block-height
- block-width
- hand-position

`block-width` *instance* [*Function*]

Arguments

instance — a `basic-block`

Return Values

an integer

Details

Returns the block's width.

The size of a block is specified as width and height, and determines which parts of the world this block occupies. No other objects can be placed to an overlapping position.

See also

- basic-block
- block-position

- block-height

`block-height` *instance* [*Function*]

Arguments

instance — a `basic-block`

Return Values

an integer

Details

Returns the block's height.

The size of a block is specified as width and height, and determines which parts of the world this block occupies. No other objects can be placed to an overlapping position.

See also

- basic-block
- block-position
- block-width

`block-supported-by` *instance* [*Function*]

Arguments

instance — a `basic-block`

Return Values

nil, or a block

Details

Returns the block this instance has been placed onto.

All blocks except for the table sit on top of another block, which supports them.

See also

- basic-block
- block-support-for

`block-support-for` *instance* [*Function*]

Arguments
instance — a `load-bearing-block`

Return Values
a list of blocks

Details
Returns the blocks that have been placed onto this instance.

See also

- load-bearing-block
- block-supported-by

1.5 Concrete block classes

In this section:

- table
- brick
- wedge
- ball

These are the blocks found in our world:

`table` [*Class*]

Superclasses

load-bearing-block , basic-block , common-lisp:standard-object, sb-pcl::slot-object, common-lisp:t

Documented Subclasses

None

Direct Slots

None

Details

The table supporting the rest of the world.

The entire rest of the world sits on this table. The table itself cannot be moved.

For each world, this class is meant to be a singleton.

Inherited Slot Access Functions

- block-support-for
- block-name
- block-width
- block-height
- block-position
- block-supported-by

`brick` [*Class*]

Superclasses

movable-block , load-bearing-block , basic-block , common-lisp:standard-object, sb-pcl::slot-object, common-lisp:t

Documented Subclasses

None

Direct Slots

None

Details

A useful movable building block with a flat top.

Because this block has a flat top, it supports other blocks.

Inherited Slot Access Functions

- block-support-for
- block-name
- block-width
- block-height
- block-position
- block-supported-by

`wedge` [*Class*]

Superclasses

movable-block , basic-block , common-lisp:standard-object, sb-pcl::slot-object, common-lisp:t

Documented Subclasses

None

Direct Slots

None

Details

An interesting movable building block.

Because this block doesn't have a flat top, it cannot support other blocks.

Inherited Slot Access Functions

- block-name
- block-width
- block-height
- block-position
- block-supported-by

`ball` [*Class*]

Superclasses

movable-block , basic-block , common-lisp:standard-object, sb-pcl::slot-object, common-lisp:t

Documented Subclasses

None

Direct Slots

None

Details

The block is a sphere.

Because this block doesn't have a flat top, it cannot support other blocks.

Inherited Slot Access Functions

- block-name
- block-width
- block-height
- block-position
- block-supported-by

1.6 The hand

In this section:

- hand
- hand-name
- hand-position
- hand-grasping

The hand is movable. It can hold at most one block.

`hand` *[Class]*

Superclasses

common-lisp:standard-object, sb-pcl::slot-object, common-lisp:t

Documented Subclasses

None

Direct Slots

grasping —

name —

position —

Details

The hand that moves the world.

This hand can be used to move every `movable-block` .

Slot Access Functions

- hand-name
- hand-position
- hand-grasping

See also

- movable-block

`hand-name` *instance* *[Function]*

Arguments

instance — a `hand`

Return Values

a symbol

Details

Returns the hand's name, a symbol.

The hand is always called `blocks-world::*hand*`.

See also

- hand

`hand-position` *instance* [*Function*]

Arguments

instance — a `hand`

Return Values

a list of two integers

Details

Returns the hand's position.

The position of a hand is specified as a list of its x and y coordinates, where the first axis runs along the table, and the second axis points upwards towards the hand.

See also

- hand
- block-position

`hand-grasping` *instance* [*Function*]

Arguments

instance — a `hand`

Return Values

a `movable-block` , or nil

Details

Returns the block the hand is currently holding.

See also

- hand
- movable-block

1.7 Other functions

`add-support` *object support* [*Function*]

Arguments

object — a `movable-block`

support — a `basic-block`

Return Values

a boolean

Details

Note that `object` has been put onto `support`.

This function maintains the slots `block-supported-by` and `block-support-for` .

See also

- movable-block
- basic-block
- block-supported-by
- block-support-for

`clear-top` *support* [*Function*]

ARGUMENTS

support — a `load-bearing-block`

RETURN VALUES

nil

DETAILS

Make space on top of this object.

Removes all blocks `support` is supporting.

SEE ALSO

- load-bearing-block
- get-rid-of
- block-support-for

`get-rid-of` *object* [*Function*]

ARGUMENTS

object — a `movable-block`

RETURN VALUES

unspecified

DETAILS

Moves `object` onto the `table` .

SEE ALSO

- movable-block
- table

- put-on

`get-space` *object support* [*Function*]

Arguments

object — a `movable-block`

support — a `basic-block`

Return Values

undocumented, but non-nil

Details

Find or make space on support for object.

See also

- movable-block
- basic-block
- find-space
- make-space

`grasp` *object* [*Function*]

Arguments

object — a `movable-block`

Return Values

t

Details

Grasps the block using the hand.

Makes sure to ungrasp the block currently grasped by the `hand` , if any.

See also

- movable-block
- hand
- ungrasp

`make-space` *object support* [*Function*]

Arguments
object — a `movable-block`
support — a `basic-block`

Return Values
undocumented, but non-nil

Details
Make space on support for object.
Takes all necessary actions to make space available.

See also

- movable-block
- basic-block
- get-space
- find-space

`move` *object support* [*Function*]

Arguments
object — a `movable-block`
support — a `load-bearing-block`

Return Values
a boolean

Details

Move block `object` onto block `support`.

This is a helper function for `put-on` .

See also

- movable-block
- load-bearing-block
- put-on

`put-on` *object support* [*Function*]

Arguments

object — a `movable-block`

support — a `basic-block`

Return Values

a boolean

Details

Move block `object` onto block `support`.

Prints the steps taken and returns T or prints an error message and returns nil.

See also

- movable-block
- basic-block
- get-space
- grasp
- move
- ungrasp

`remove-support` *object* [*Function*]

Arguments

object — a `movable-block`

Return Values

a boolean

Details

Note that `object` has been taken from `support`.

This function maintains the slots `block-supported-by` and `block-support-for`
.

See also

- movable-block
- block-supported-by
- block-support-for

`ungrasp` *object* [*Function*]

Arguments

object — a `movable-block`

Return Values

a boolean

Details

Ungrasps the block if hand is holding it.

Returns t if successful, or nil if the `hand` didn't hold this block.

See also

- movable-block
- hand
- grasp

Chapter 2

The blocks-world-goals package

This package contains the source code of chapter 22, *"Answering Questions about Goals"* from Lisp (3rd edition) by Winston and Horn.

2.1 Lots of undocumented functions

I was too lazy to document this package, which is why all its functions have a big fat "undocumented" warning.

This package's page also looks rather empty and sad.

2.2 Other functions

`attach-action` *node action* [*Function*]

No documentation string. Possibly unimplemented or incomplete.

`attach-parent` *child parent* [*Function*]

No documentation string. Possibly unimplemented or incomplete.

`find-action` *given-form* `&optional (`*node* `*current-node*)` [*Function*]

No documentation string. Possibly unimplemented or incomplete.

`node-action` *object* [*Function*]

No documentation string. Possibly unimplemented or incomplete.

`node-children` *object* [*Function*]

No documentation string. Possibly unimplemented or incomplete.

`node-parent` *object* [*Function*]

No documentation string. Possibly unimplemented or incomplete.

`show-simple-tree` *node* `&optional (`*indentation* `0)` [*Function*]

No documentation string. Possibly unimplemented or incomplete.

2.3 Other macros

`define-history-method` *name parameters &rest body* [*Macro*]

No documentation string. Possibly unimplemented or incomplete.

`tell-why` *name &rest parameters* [*Macro*]

No documentation string. Possibly unimplemented or incomplete.

2.4 Other classes

`node` *[Class]*

SUPERCLASSES

common-lisp:standard-object, sb-pcl::slot-object, common-lisp:t

DOCUMENTED SUBCLASSES

None

DIRECT SLOTS

action —

children —

parent —

No documentation string. Possibly unimplemented or incomplete.

2.5 Other variables

`*current-node*` *[Variable]*

No documentation string. Possibly unimplemented or incomplete.

Index