CS 251 Program 1 *Blocks*: The Block Engine

Due: Thu Feb 10, 2000 5PM **Spec version: 0.8**

Introduction

The goals of the first project are to work with classes and objects and arrays, to get some experience designing and building multiple file programs, and to understand a bit about reading, understanding, and implementing from specifications. We will implement a program called 'blocks' that simulates a simple kind of 'physics' for blocks sliding around on a gridded board.

Your job in this first program is to implement a blocks 'engine' that operates according to a specific set of commands and responses. An *engine*, informally speaking, is a program or some code that is designed to work behind the scenes or under the hood, rather than interacting directly with a human user. Instead, other programs or code will sit between the engine and the user, playing the role of the steering wheel and gas pedal and tachometer and so forth.

It is of critical importance, obviously, that the engine and the gas pedal agree on exactly what messages can be sent and what they will mean. A *communications protocol* provides a set of rules, describing who can say what to whom when and in what way, that all mutually communicating parties must agree on. Below, find the description of the ***Blocks*** protocol that your program must implement.

The Blocks Protocol

General protocol issues:

- Each protocol message in either direction are terminated with a single newline (i.e., the thing that `'endl'` generates in C++). The protocol consists of a series of *interactions*. Each interaction with the engine begins with a command to the engine followed by zero or more responses from the engine.
- The *only* thing the engine outputs is protocol responses. That implies, specifically:
    1. Your program must output ***nothing*** when it first starts up.
    2. Your program must ***not*** output any 'debugging info' while it runs. If you put in any 'debugging print statements' during development, you ***must*** be sure to remove them all before turning in your program, or it will fail testing and be marked down.
    3. Your program must ***not*** output any 'prompts' such as 'Enter command:' or anything of the sort, or it will fail testing and be marked down.
- Your program *may*, but is not required to, accept lower−case forms of the commands described below. Your program *must* generate all responses in upper−case as described below. There is no grading penalty for failing to accept lower−case commands, and no grading bonus for doing so.

- All response arguments are separated by single space characters. There is no leading or trailing spaces in responses. There is a single newline ending each response.
- The engine may deal with illegal commands and/or illegal command arguments in any way that it wants to provided that the program is not at risk of crashing. In particular, the engine *may* simply exit if a command or command error occurs, or *may* ignore the error and attempt to continue functioning, or *may* give an error message and attempt to continue functioning. It *must not* continue functioning if further execution would be unsafe.

Commands the engine reads from cin

`A`	engine must produce an **A**UTHOR response containing the name of the author of the engine
`D` *`milliseconds`*	engine must produce a **D**ELAY *milliseconds* (int >=0) response (Note the engine does ***NOT*** actually perform a delay or waste time here! It just outputs the DELAY response!)
`I` *`width height blockcount`*	**I**nitialize the board to have *width* (int 1:50) and *height* (int 1:50), with *blockcount* (int 1:100) blocks
`B` *`num x y movable`*	set **B**lock *num* (int 0:blockcount−1) initial position to *x* (int 0:width−1), *y* (int 0:height−1), and specify whether it is *movable* (int 0:1).
`R`	**R**eset all blocks to their initial positions, generating first a RESET response and then blockcount BLOCK responses
`P` *`num direction`*	**P**ush block *num* (int 0:blockcount−1) in given direction (char 'N', 'S', 'E', 'W'). Engine generates a possibly empty sequence of MOVE, PUSHED, and HALT responses until all the effects of the Push have died out
`Q`	engine generates a QUIT response and then **Q**uits

Responses the engine writes to cout

`RESET` *`width height`*	Board set to size *width* (int 1:50) by *height* (int 1:50).
`AUTHOR "`*`authorname`*`"`	Author of the program is *authorname*. (The *authorname* can contain any characters except for newlines and double quotes, and is surrounded by double quotes in the AUTHOR response.)

`BLOCK` *`num x y movable`*	Block *num* (int 0:blockcount−1) has been placed at *x* (int 0:width−1), y (int 0:height−1) and can be moved if *movable* (int 0:1)
`MOVE` *`num x y`*	Block *num* (int 0:blockcount−1) has moved from its previous location to *x* (int 0:width−1), *y* (int 0:height−1).
`DELAY` *`milliseconds`*	A delay of *milliseconds* (int >=0) milliseconds should occur here. (Note the engine does ***NOT*** actually *do* any delaying itself!)
`PUSHED` *`num`*	Block *num* (int 0:blockcount−1) has been pushed
`HALT` *`num`*	Block *num* (int 0:blockcount−1) has halted
`QUIT`	Engine is about to quit.

Board semantics

The board has a current width and current height, each of which may be as small as 1 or as large as 50. The current width and current height are established when a 'I' command is processed, and may be changed within a single run of the engine if further 'I' commands are entered.

For purposes of illustration here, the coordinate system is this: increasing *x* moves left to right, and increasing *y* moves top to bottom. So a width=3, height=2 board would have these *(x,y)* coordinates:

(0,0)	(1,0)	(2,0)
(0,1)	(1,1)	(2,1)

When the engine is first started, before any commands have been read, the system is in a state as if an '`I 1 1 1`' command had been executed.

Block semantics

An 'I' command also establishes (or changes, if it's not the first 'I' command) the *blockcount*, which is number of blocks on the board. The *blockcount* may be as small as 1 and as large as 100. After an 'I' command, the *blockcount* blocks, which are numbered ranging from 0 up to *blockcount*−1, all have the following properties: They are located at coordinates (0,0) and they are movable, but are currently halted.

The heart of the blocks program is its response to a 'P' command, in which a block is pushed in one of the four compass directions. In response, that block may move zero, one, or many times, and may bump into other blocks, possibly causing them to move.

Several examples here will help clarify how the engine operates. After the examples are some more details about the 'settling' process that occurs when a 'P' command is processed.

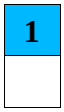
In these examples, note that each command is a *single line* of input, even if it appears on multiple lines due to document formatting. Note that there may zero, one, or more than one responses to each command. Note, likewise, that each response (of the possibly

generated by a single command) is a *single line* even if it appears on multiple lines due to line breaking in this document.

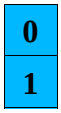
Example 0

In this first example, a 1x2 board is created containing two blocks, and then one block is pushed. No 'B' commands are given so the blocks start in the default position as discussed above.

For reference, the starting configuration for this example is:



where in fact both block 0 and block 1 are in (0,0). The ending configuration is



*In all these illustrations, note that x increases moving to the **right**, and y increases moving **down**!*

Command to engine	*Response(s) from engine*	*Notes*
`I 1 2 2`		*width 1, height 2, blockcount 2*

`R`	`RESET 1 2`
	`BLOCK 0 0 0 1`
	`BLOCK 1 0 0 1`
		When a reset command is received, RESET and then BLOCK responses are generated

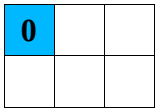
`P 1 S`	`PUSHED 1`
	`MOVE 1 0 1`
	`HALT 1`
		Block 1 is pushed, moves south, hits the wall, and halts

`Q`	`QUIT`	*Then the engine exits..*

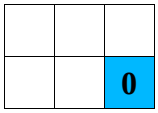
Example 1

In this example, a 3x2 board is created containing one block, and then the block is pushed around twice. This shows the basic mechanics of the engine, but not any of the interactions that can happen *between* blocks.

For reference, the starting configuration for this example is:



and the ending configuration is



*In all these illustrations, note that x increases moving to the **right**, and y increases moving **down**!*

Command to engine	*Response(s) from engine*	*Notes*
`I 3 2 1`		*width 3, height 2, blockcount 1*
`B 0 0 0 1`		*block 0 is at (0,0) and is movable*
`R`	`RESET 3 2` `BLOCK 0 0 0 1`	*When a reset command is received, RESET and then BLOCK responses are generated*

`P 0 S`	`PUSHED 0`
	`MOVE 0 0 1`
	`HALT 0`
		Block 0 is pushed, moves south, hits the wall, and halts

`P 0 E`	`PUSHED 0`
	`MOVE 0 1 1`
	`MOVE 0 2 1`
	`HALT 0`
		Block 0 is pushed. moves east twice, hits the wall, and halts

`Q`	`QUIT`	*Then the engine exits..*

Example 2

This example shows both fixed and movable blocks, and shows blocks banging into each other.

For reference, the starting configuration for this example is:

and the ending configuration is

Command to engine	*Response(s) from engine*	*Notes*
`A`	`AUTHOR "Dave Ackley"`	*your engine's response will be different!*
`I 6 1 3`		*width 6, height 1, blockcount 3*
`B 2 4 0 1`		*block 2 is at (4,0) and is movable*
`B 0 3 0 1`		*block 0 is at (3,0) and is movable*
`B 1 1 0 0`		*block 1 is at (1,0) and is fixed*

`R`	`RESET 6 1`
	`BLOCK 0 3 0 1`
	`BLOCK 1 1 0 0`
	`BLOCK 2 4 0 1`
		When a reset command is received, RESET and then BLOCK responses are generated

`P 1 S`		*Block 1 is fixed, no response*
`P 0 N`	`PUSHED 0` `HALT 0`	*Block 0 is movable, but there's no place to move north*

Command to engine	*Response(s) from engine*	*Notes*

`P 0 W`	`PUSHED 0`
	`MOVE 0 2 0`
	`HALT 0`
		Block 0 is pushed west. It moves one square, runs into fixed block 1, and halts

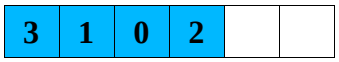
`P 0 E`	`PUSHED 0`
	`MOVE 0 3 0`
	`PUSHED 2`
	`HALT 0`
	`MOVE 2 5 0`
	`HALT 2`
		Block 0 is pushed east. It moves one square (back to where it started), then bumps into block 2 and halts. Block 2 moves one square then hits the edge and halts.

`D 3000`	`DELAY 3000`	*A three second delay is requested*
`Q`	`QUIT`	

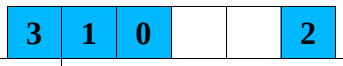
Example 3

This final example shows how ‘pushing’ can be transmitted through multiple blocks, and how multiple ’r’ commands can be given.

For reference, the starting configuration for this example is:



and the configuration just before the second ’R’ command is



Command to engine	*Response(s) from engine*	*Notes*
`I 6 1 4`		*width 6, height 1, blockcount 4*
`B 3 0 0 1`		*block 3 is at (0,0) and is movable*
`B 1 1 0 1`		*block 1 is at (1,0) and is movable*
`B 0 2 0 1`		*block 0 is at (2,0) and is movable*
`B 2 3 0 1`		*block 2 is at (3,0) and is movable*

`R`	`RESET 6 1`
	`BLOCK 0 2 0 1`
	`BLOCK 1 1 0 0`
	`BLOCK 2 3 0 1`
	`BLOCK 3 0 0 1`
		When a reset command is received, RESET and then BLOCK responses are generated

`P 3 E`	`PUSHED 3`
	`PUSHED 1`
	`HALT 3`
	`PUSHED 0`
	`HALT 1`
	`PUSHED 2`
	`HALT 0`
	`MOVE 2 4 0`
	`MOVE 2 5 0`
	`HALT 2`
		Block 3 is pushed east. It pushes block 1 and then halts. Block 1 pushes block 0 and then halts. Block 0 pushes block 2 and then halts. Block 2 moves two squares east then hits the wall and halts.

Command to engine	*Response(s) from engine*	*Notes*
R	RESET 6 1 BLOCK 0 2 0 1 BLOCK 1 1 0 0 BLOCK 2 3 0 1 BLOCK 3 0 0 1	*When a reset command is received, RESET and then BLOCK responses are generated. Not much point in resetting just before quitting, but it's legal.*
Q	QUIT	

'Settling' the board

As the examples above show, a 'P' command can cause a long sequence of responses to be generated. For your blocks program to be correct, it must generate the responses in a specific order. In broad strokes, here is a scheme that generates responses in the correct order and with reasonable efficiency.

- Each block keeps track of its 'state', so that it can tell whether it is currently moving in some direction, or is movable but currently halted, or is not movable.
- When a 'P' command occurs, the board tells the relevant block it has been pushed in the given direction; if the block is movable, it notes that it has been pushed and changes its current direction.
- The board then goes through all the blocks in increasing order, giving each one in turn the option to move. Each block reports back whether it moved or changed its state. If *any* block says it did move or change state, then after going through all the blocks, the board repeats this whole step.
- Otherwise, if on a complete pass through the blocks, they all reported that they didn't move or change state, then the 'settling' process is completed.

You'll need to fill in the details of this process, such as precisely where and when to generate which responses, as your design develops.

Engine limits

The following limits apply to the blocks engine:

1. The maximum number of blocks in use is 100.
2. The minimum number of blocks in use is 1.
3. The maximum size of the board is 50x50.
4. The minimum size of the board is 1x1.
5. All commands except 'P' must complete in no more than O(blockcount) time. 'P' commands must complete in no more than *k**O(blockcount) time, where *k* is the total number of MOVE, HALT, and PUSHED responses generated before the system settles.

Revision History

- Version 0.8, first released version,1/25/00. All the major pieces are in place anyway.