Report

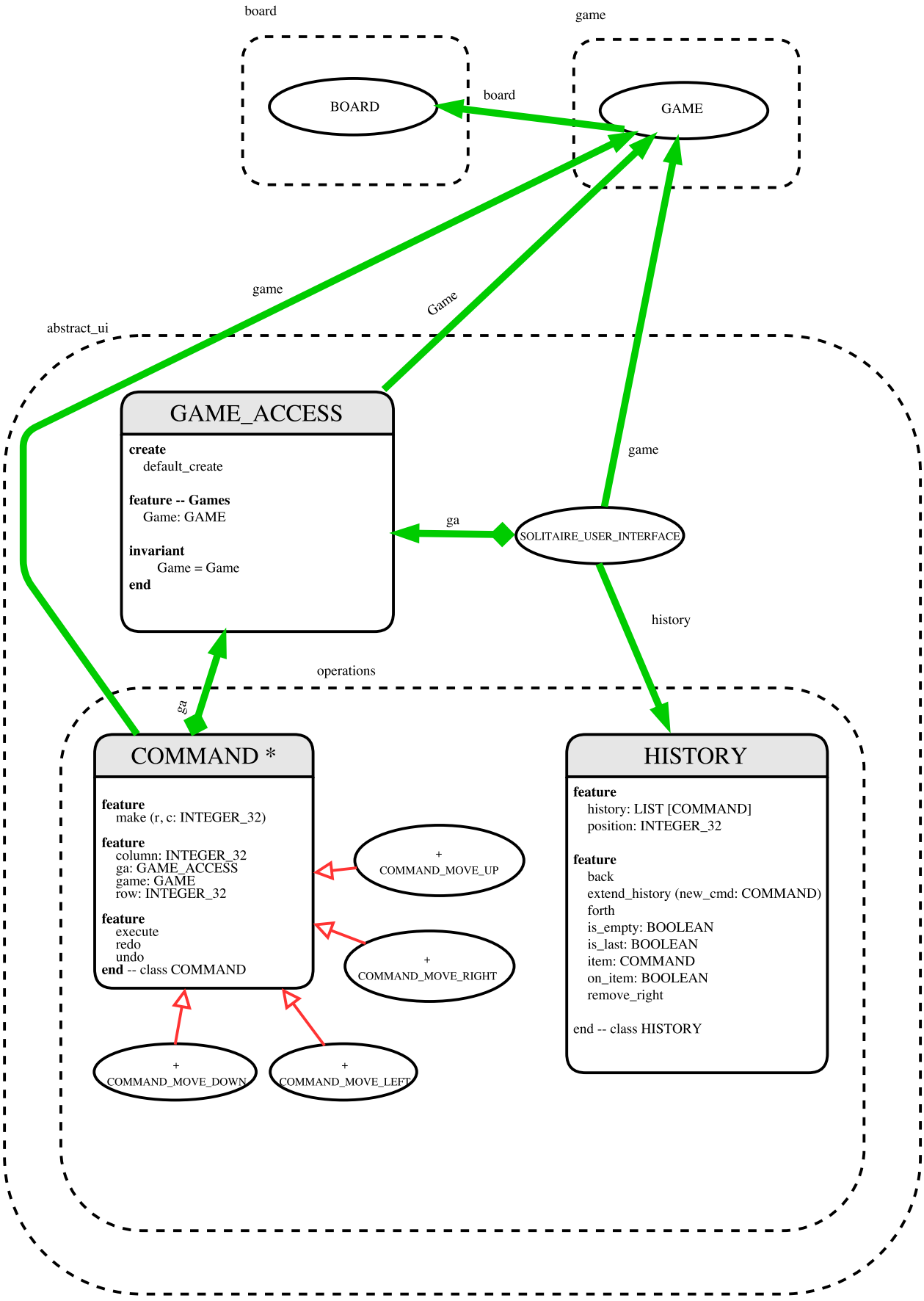
Lab 4: Peg Solitaire with Undo and Redo

EECS 3311

Fall 2017

Saad Saeed

saeeds28


board
game
BOARD
board
GAME
game
Game
abstract_ui
GAME_ACCESS
create
default_create
feature -- Games
Game: GAME
invariant
Game = Game
end
game
ga
SOLITAIRE_USER_INTERFACE
history
operations
ga
COMMAND *
feature
make (r, c: INTEGER_32)
feature
column: INTEGER_32
ga: GAME_ACCESS
game: GAME
row: INTEGER_32
feature
execute
redo
undo
end -- class COMMAND
+
COMMAND_MOVE_UP
+
COMMAND_MOVE_RIGHT
+
COMMAND_MOVE_DOWN
+
COMMAND_MOVE_LEFT
HISTORY
feature
history: LIST [COMMAND]
position: INTEGER_32
feature
back
extend_history (new_cmd: COMMAND)
forth
is_empty: BOOLEAN
is_last: BOOLEAN
item: COMMAND
on_item: BOOLEAN
remove_right
end -- class HISTORY

The undo and redo features for this lab were implemented via the state pattern. A single deferred class COMMAND had all the common method signatures (execute, undo, and redo) that all its descendant classes implemented. These classes were COMMAND_MOVE UP, DOWN LEFT, RIGHT. A class HISTORY was used to store the move objects-via an ARRAYED_LIST object of type COMMAND-for undo and redo purposes. Whenever a new move was made within a SOLITAIRE_USER_INTERFACE class, an object of that move type was created and stored at the end of the ARRAYED_LIST. This was achieved by deleting the elements to the right of the current element pointed by the iteration cursor. Furthermore, moving the iteration cursor back for undo and forward for redo called the undo and redo feature of the particular move pointed by it. Dynamic binding comes into play when deciding whose undo and redo features to call. This makes the code more dynamic and extremely reusable because if a new type of move was added, all we would have to do is inherit from the COMMAND class and implement all the inherited features. Nothing else in the code would change.

A singleton pattern was maintained by changing the do of the GAME_ACCESS class to a once, so that only one board would get created.