Sohrab Oryakhel
214302145

Tic Tac Toe Game Implemented Using Command Pattern

TABLE OF CONTENTS

Design View: Bon class Diagrams for Model Cluster

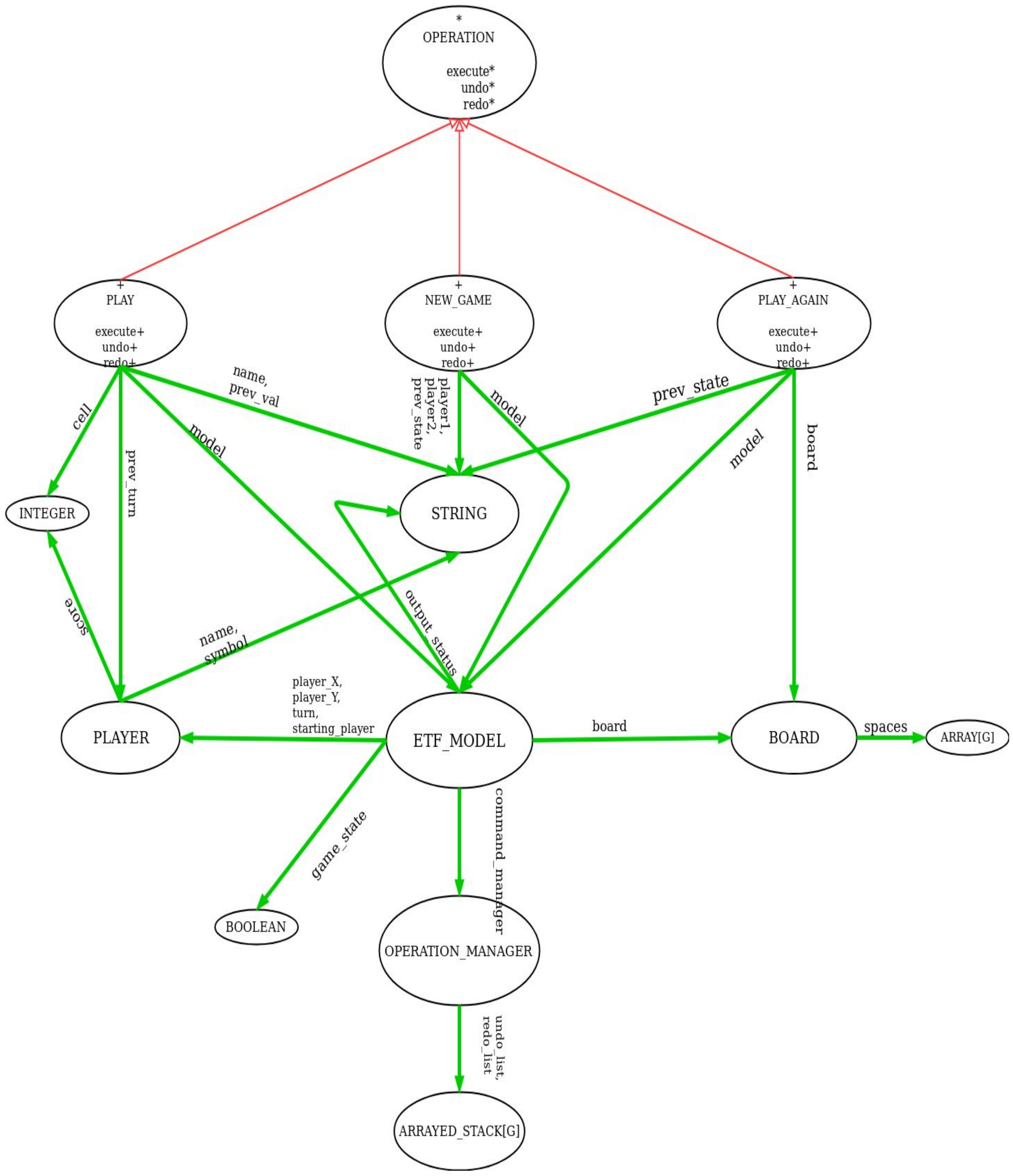


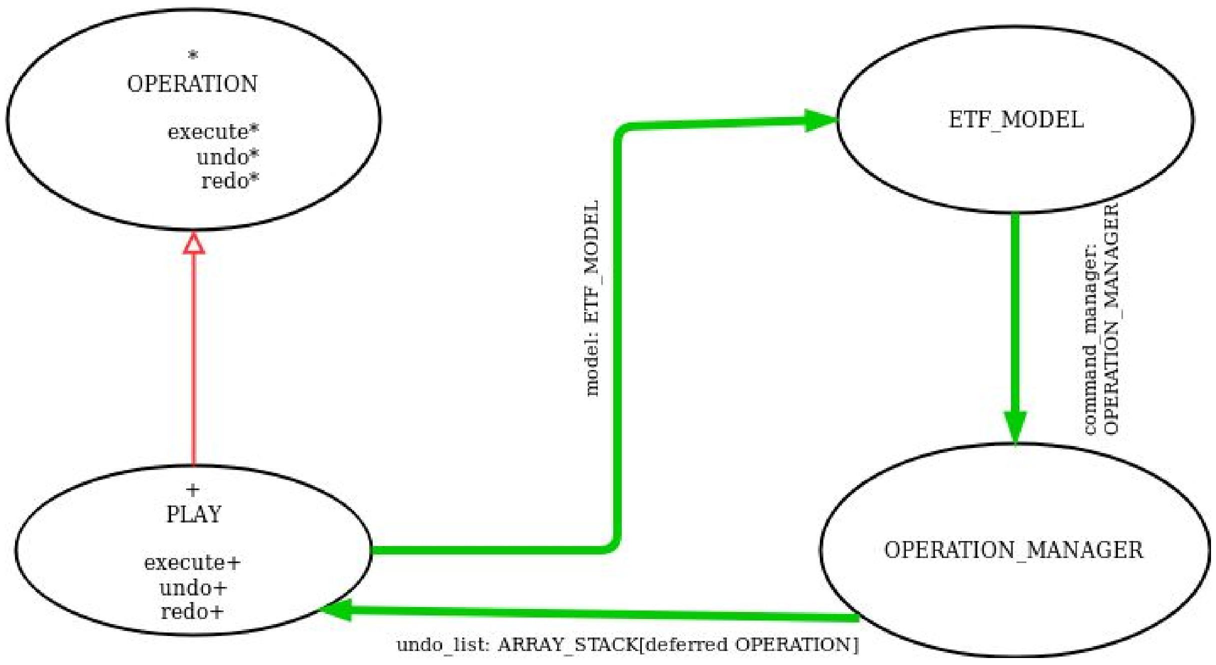
Table of Significant Modules

OPERATION	Deferred or abstract class representing the main operations of the program: - execute - undo - redo The above operations are deferred.
OPERATION_MANAGER	Manager class of the execute, undo/redo operations and also the container of the undo and redo lists which are implemented using a STACK. Commands are executed as instances of OPERATION in this class and are thereafter added to the undo list. Undo and redo lists, as stacks which allow for simplicity, remove their items once undone or redone. This way the lists remain small. Also because these lists are in this class, undo and redo do not need to exist as separate commands.
NEW_GAME	Inherits from OPERATION and redefines execute, undo and redo. This class has an instance of the model class, ETF_MODEL, which is the main class. This is to change the state of the main class accordingly once the command is made. An object of this type will be in the undo/redo lists of command manager In execution, this command first checks the inputs for errors. If there are no errors then the command effectively changes the state of the model. Otherwise it gives error messages to model. Undo sets model to its previous state (previous output) and redo runs the execute routine.
PLAY	Inherits from OPERATION just like NEW_GAME. Also has an instance of model for state changing.

	In execution, this command first checks the inputs for errors. If there are no errors then the command effectively changes the state of the model. Otherwise it gives error messages to model. Undo sets model to its previous state (previous output) and redo runs the execute routine.
PLAY_AGAIN	Same Inheritance as PLAY and NEW_GAME. In execution, this command first checks the inputs for errors. If there are no errors then the command effectively changes the state of the model. Otherwise it gives error messages to model. Undo sets model to its previous state (previous output) and redo runs the execute routine.
PLAYER	A class representing a player (could be either player X or O). Has the name, symbol and score for each player, and also setter and update commands.
BOARD	This is the game board. A tic tac toe board represented with an array of nine spaces. There are additional methods to check the state of the board.
ETF_MODEL	This is the main game class. Everything happens form here. This class has a BOARD, two PLAYERs, and an OPERATION_MANAGER. It has routines for commands which are created and passed onto the command manager and executed, added to the undo/redo list, etc. Undo and redo are routines defined in OPERATION_MANAGER which are called through this class's OPERATION_MANAGER instance. All other commands are called from this class initially.

Detecting a winning game

A game is won if the last move a player made resulted in a winning configuration for that player. A winning game always follows a play command. This means a player must run *play(name, button)* which will change the state of model and hence the state of the game board. This happens by the play command putting the player's symbol in place of the button in the game board. When the state of the board is changed, the PLAY object invokes the model's *update_score* routine which checks to see if the board is in a winning configuration. It does this check by running the *is_player_x_winner* or the *is_player_o_winner* routines of the game board. If the board is in a winning configuration then that player's score is updated (previous score plus one).



Undo/Redo Design

The undo/redo design is such that *undo* and *redo* are commands made by the user which get called as routines first in the model, the command_manager, the actual command instance. Game commands like *play, new_game,* and *play_again* all inherit from OPERATION. Meaning they will exists as instances of OPERATION once they are called. They will also have their own execute, undo, and redo routines. Once these commands are executed they will be popped onto the *undo_list* in OPERATION_MANAGER, the manager class of the lists. This means that *undo* and *redo* are not instances of OPERATION or they do not inherit from it which is simply because they don't need to. *Undo and redo* are not OPERATIONs, they are only routines in OPERATION_MANAGER. The crucial thing is that once the commands are in the *undo_list*, the *undo* command will be a routine in OPERATION_MANAGER which will run the the *undo* routine in the actual command instance. After this the *undo* routine in OPERATION_MANAGER will put the undone command in the *redo_list* and that list works the same as its counterpart.