Tic Tac Toe

Name: Bryan Brkic
Student Number: 213958806
Prism Login: brkicb

Signature: 

Table of Contents

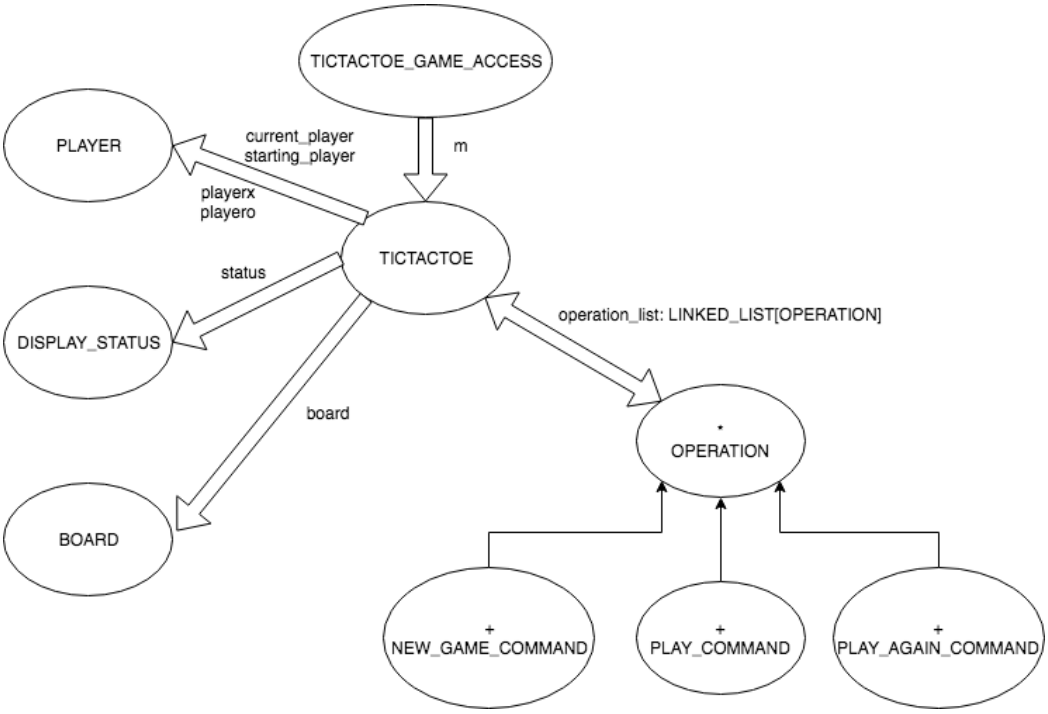
1. Top level view

Tictactoe is a game where two players take turns playing on a three by three board of squares, placing an X and O on a board square. The first player places an X on a square, then the next player an O, and they keep switching turns playing. If a player has three of their symbols on the board in a row, whether its down, across, or diagonal, that player wins the game. If neither player has three of their symbols in a row, the game is a tie. If the players play again, they switch who makes the first play.

The ETF grammar for the tictactoe game shown below demonstrates what commands the user can execute. As a user, we have access to the new_game command that would take two arguments, the first being the name of the first player and the second being the name of the second player. Then we have the play command, which also takes two arguments, the first being the name of the player to make a move on the game board, and the second is the position on the game board to place the player's symbol (X or O). Then we have the play_again command which allows players to play the game again only if they finished their current game, and when they play again, the starting player switches. Then we have the undo command which undos a play command, and also if you had an error during a play like trying to play on an occupied position, by using undo you can see these error messages appear again which doesn't change the state of the board. So if there were errors and the game played on, undo won't always change the board state but sometimes the message state. Then the redo command redos what you last undid, and if there is no undo left to redo, redo won't do anything.

```
type NAME = STRING
type BUTTON = 1..9

new_game(player1: NAME; player2: NAME)
 — add players `player1' and `player2'
 — `player1' starts X

play(player: NAME; press: BUTTON)

play_again

undo
 — last action while in play
 — otherwise no effect

redo
 — last action while in play
 — otherwise no effect
```

BON class diagram overview (*architecture* of the design)



The TICTACTOE class represents a game of tictactoe, and the user can only obtain one instance of TICTACTOE using TICTACTOE_GAME_ACCESS through a singleton design pattern. The PLAYER class is used to represent one player during the game, where they have assigned a name, symbol (X or O), and a score that increases when they win a game. The DISPLAY_STATUS class is used by TICTACTOE to have access to various status messages that can be displayed to the user and keeps track of the current status and current message. The BOARD class is used for representing the state of the game board. In this class you have access to procedures for placing a symbol on the board, and you have access to functions for printing the current board state, and also for checking the current state of the board. For checking the current state of the board, since there a four possible states (X won, O won, game still in progress, or game ended in a tie), we have the function return an integer from 1 to 4 where each of these integers represents one of these states, and we can use this integer to determine what the current state of the board is. Then we have the operations that the user can perform. Whenever an operation is performed, TICTACTOE stores the operation in a linked list where you can then undo by going back in the linked list to be in the previous state, or redo by going forward in the linked list to be in the next state if there is one. We have NEW_GAME_COMMAND which handles creating a new game and clears the linked list if there is no error. PLAY_COMMAND is used for handling when a user makes a play and handles what happens when a user makes an error like trying to play on an occupied square, and also handled what happens when a player wants to undo or redo a play. Then PLAY_AGAIN_COMMAND is used for allowing players to play a game again if the game has finished and clears the linked list if there are no errors in running the command.

2. Table of module - responsibilities and secrets

1.1.1	TICTACTOE	**Responsibility**: Represents the current state of a tictactoe game.	**Alternative**: none
	Concrete	**Secret**: Includes information about player x and player o, who the current player is, which player started the game, the game over status, the board, the display status messages, current status, current message, and the operations, which get stored in a linked list.	

1.1.1	PLAYER	**Responsibility**: Represents a player during a game, whether it's the player with symbol X or O.	**Alternative**: none
	Concrete	**Secret**: none	

1.1.1	BOARD	**Responsibility**: Represents the state of the board and allows for placing an X or an O on a specific square. Also gives you information on what the state of the board is. This includes the location of symbols placed on the board by being able to print the board, and also information on the state of the game, so whether player x won, player o won, if the game is in progress, or if the game ended in a tie.	**Alternative**: none
	Concrete	**Secret**: An array of size 9 that stores strings is used to represent the board. An underscore in the array is an empty square, and an X is a play made by player x and an O is a play made by player o.	

1.1.1	DISPLAY_STATUS	**Responsibility**: Where you have access to different possible display messages that are used, and keeps track of the current status and the current message.	**Alternative**: none
	Concrete	**Secret**: none	

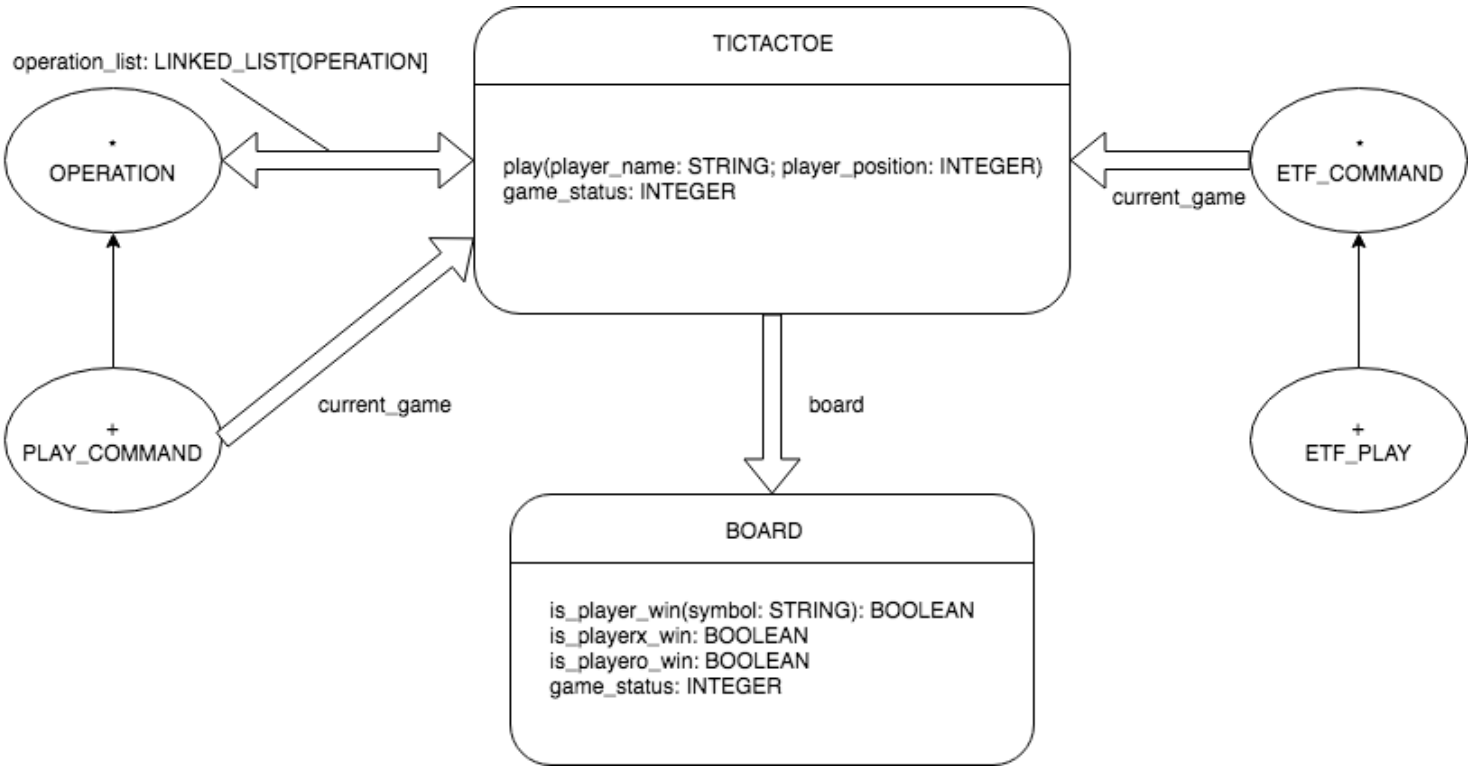
1.1.1	OPERATION	**Responsibility**: Represents an operation in a tictactoe game.	**Alternative**: none
	Abstract	**Secret**: none	

1.1.1	NEW_GAME_COMMAND	**Responsibility**: Operation used for creating a new tictactoe game to be played.	**Alternative**: none
	Concrete	**Secret**: none	

1.1.1	PLAY_COMMAND	**Responsibility**: Operation used for a player to place their symbol (X or O) on the board.	**Alternative**: none
	Concrete	**Secret**: none	

1.1.1	PLAY_AGAIN_COMMAND	**Responsibility**: Operation used for allowing two players to play a new game if the two players have finished the game they were playing.	**Alternative**: none
	Concrete	**Secret**: none	

3. Detecting a winning game

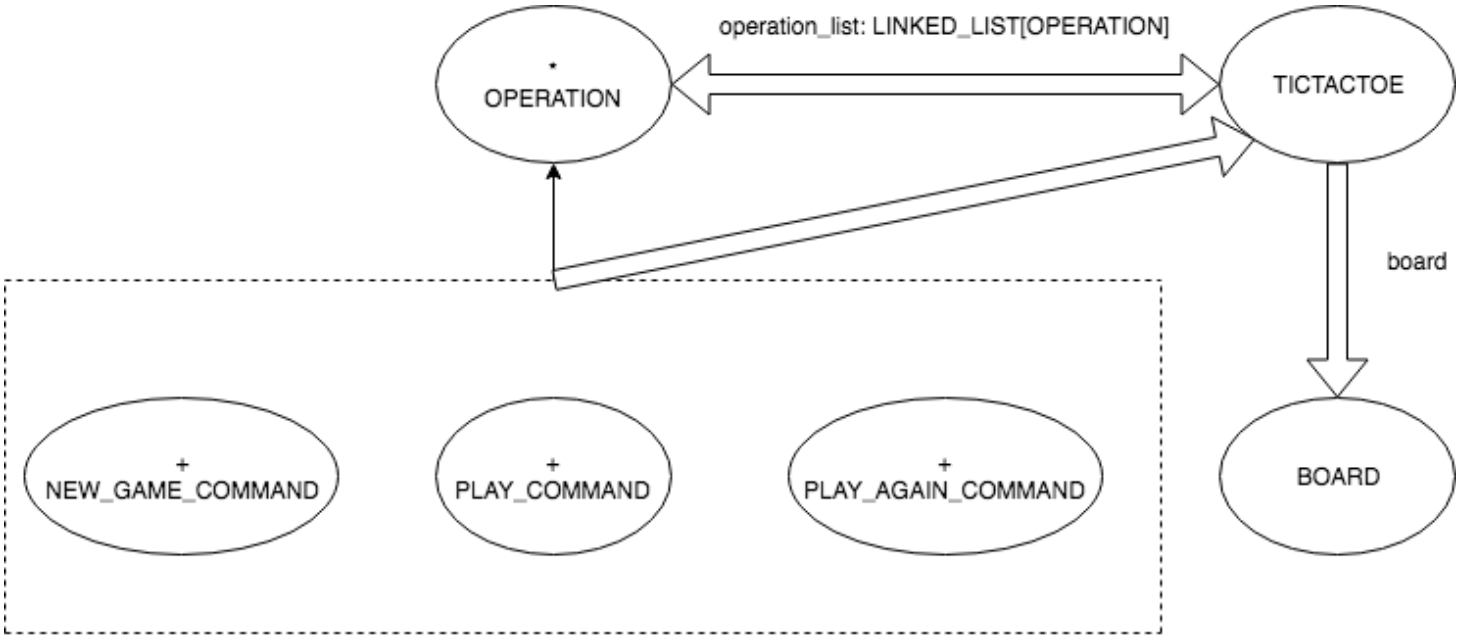


In order to detect a winning game, the TICTACTOE class uses a function called game_status to check what the current status of the game is. This game_status function uses the BOARD class's game_status function to receive an integer value that represents the state of the board which the TICTACTOE class uses to know what the current state of the game is. The game_status function in the BOARD class returns a 1 if playerx won, 2 if playero won, 3 if the game is still in progress, and 4 if the game ended in a tie.

When a play is made, PLAY_COMMAND checks the status of the game right after the play is made by using the game_status function in TICTACTOE which uses the game_status function in BOARD to see what the status is. Therefore, to detect a winning game, either a 1 will be returned by game_status to show that playerx won the game, or a 2 will be returned by game_status to show that playero won the game. Then after it's determined who won the game, the game_over_status is TICTACTOE is set to true, the status and message are updated accordingly and the appropriate players' score is increased. Then the operation_list attribute which is a linked list in TICTACTOE that holds a list of operations performed is cleared so that undo and redo won't perform any action. This way the only options for the user is to either start a new game or to play again.

The PLAY_COMMAND class also will only ever check the game status if the play that was made was a valid play, if its detected that the play was invalid, then instead of checking the state of the game and making any changes to it, all that is updated are the status and message displayed to the user.

4. The undo/redo design



To implement the undo/redo design into the tictactoe game, I made a deferred class OPERATION which has one attribute I added called error_present, which is a boolean value used for checking if there was an error. Then there are three deferred features; execute, undo, and redo. NEW_GAME_COMMAND, PLAY_COMMAND, and PLAY_AGAIN_COMMAND all inherit OPERATION and thus have access to the error_present attribute, and they all need to add definitions for execute, undo and redo.

In order for undo and redo to work, the TICTACTOE class needs a way of storing the operations performed. To store operations, I used a LINKED_LIST that takes OPERATION as its elements.

Undo and redo are mainly used for undoing and redoing play commands. Undoing and redoing doesn't always change the board state but instead sometimes shows a previous status and message that was present when an operation was attempted.

After using the new_game or play_again command, you don't want to be able to perform undo and redo commands. However if during a game you try to make a new_game but an error occurs, then if you were to do a play command (that doesn't end the game) then undo, you would see the status and message that showed up when you incorrectly used the new_game command. Same goes for the play_again command.

I had the undo command in the NEW_GAME_COMMAND class set the status and message to the old status and message that was present. In the TICTACTOE class the undo command there checks for if the NEW_GAME_COMMAND is the first operation in the LINKED_LIST and if we are currently at the first element, that way all that happens is that an ok status is sent. Then there is also a check for if we currently are not at the first element, but there is a NEW_GAME_COMMAND as the first element in the LINKED_LIST with an error present. This way if someone were to start off by running new_game with an error, then ran new_game again with an error, then did an undo and redo command, the program would be able to display correctly to the user.

For the PLAY_AGAIN_COMMAND class, the undo and redo is handled very similarly. All undo does in PLAY_AGAIN_COMMAND is set the status and message to the old status and message and redo just runs execute. Then in the TICTACTOE class, the undo command has a check for if PLAY_AGAIN_COMMAND is the first element in the LINKED_LIST and whether we are currently at the first element in the LINKED_LIST, that way if that is the case and an undo command is run, the state of the game shouldn't change.

Finally, we have undo and redo for the PLAY_COMMAND class. For the PLAY_COMMAND class, undo simply check if there was an error present when the play command was run, if so, the board state doesn't change but all that happens is that the status and message are set to the old status and message. If there was no error then you have to check if the game is not over, and if that is the case then the position of the square where the player played last is set to be empty, the status and message are set to the old status and message, and the players switch turns.

The redo command for all operations is the same. In the TICTACTOE class, when the redo command is run, if there is no valid cursor position to the left of the cursor (meaning that we are in the first position), then you don't do anything. Otherwise, you check if you're not in the first or last position, and if that's the case, you go forth in the LINKED_LIST and run redo which in all operations just runs the execute command for the specific operation.