Assessment – 3

Final Design and report Implementation

Name- Aditya Venkataraman Iyer

Student number- N10555650

Subject- Object-oriented design

Statement of Completion:

The final design and implementation are one of the most interesting and enthralling projects for me. The requirements were very clear and not much interpretation was required for this project.

The requirements that were completed in the Implementation are

- The project demanded an extensive framework for 3 board games- Tic tac toe, Reversi and Nine Men Morris, with at least two games in the Implementation. I was able to implement Tic tac toe and Nine Men Morris board Game.
- Further going deep in the game, there should be an option for the user to play either Vs Friend or Computer algorithm. I was able to complete both the options for the two games mentioned above.
- The validity of the moves is one of the most important features for any game and I was able to complete the Validity of the player moves and the AI for both the games.

The requirements that were left half-open

- The requirement stated range of option for the player to play against AI. The least two strategies that were expected were Alpha Beta algorithm and Random. Unfortunately, I was able to complete only the Random strategy.
- There should be a primitive Online help system that allows users to users with the Game. Unfortunately, I was able to only give the weblinks and the not the complete user guide and the assistance to the users.
- The users should be able to save the game and also undo the last move played. The game implantation allows users only to save the game but not open it back again and also doesn't allow undoing any move.

Although the implementation doesn't allow for some of the listed requirements the design completes all the requirements.

Final Design

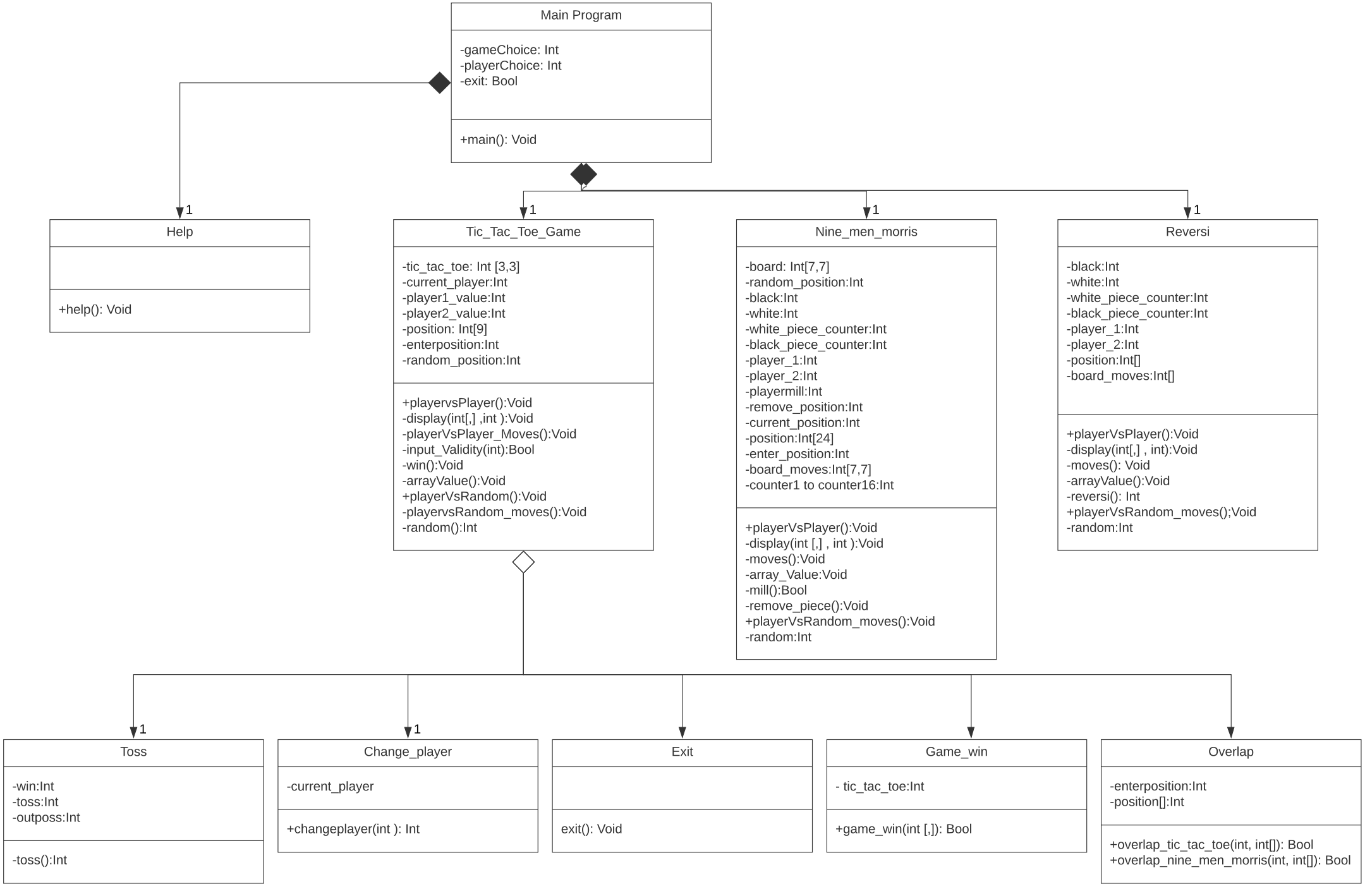
The design is done so that there is an extensive frame that is reusable for any further enhancement in future. The design is created with the help of two major classes that are derived from the Main class. There are 10 classes in all with the Main class being the entry point. The three games are divided into three namely Tic tac toe, Reversi and Nine Men Morris. The Primitive Help system is divided into a separate class so that the further enhancement of the moves and the Hints which can be an added advantage to the player can be included. Toss is divided into a separate class as it has to be one of the common features for all three games that can be further enhanced with the use of GUI. Change player is also separated into different class as the similarity for the code increases every time the code is reentered. Storage class although not been implemented is one of the most class in the design as the class holds the moves that player plays. This storage of moves is really helpful later to undo a move or load a saved game. Game win as the name suggests is the class which determines the final winner of the game. One of the main reason why the Game win has been a separate class is to improve the game condition and simplicity. The game allows the user to store the game moves and position and it would be very difficult to have a system that has the same class as to play and also declare a winner when the moves are undone, hence this class is separated. Exit class is just a basic exit at any point when the user wants hence increasing the flexibility.

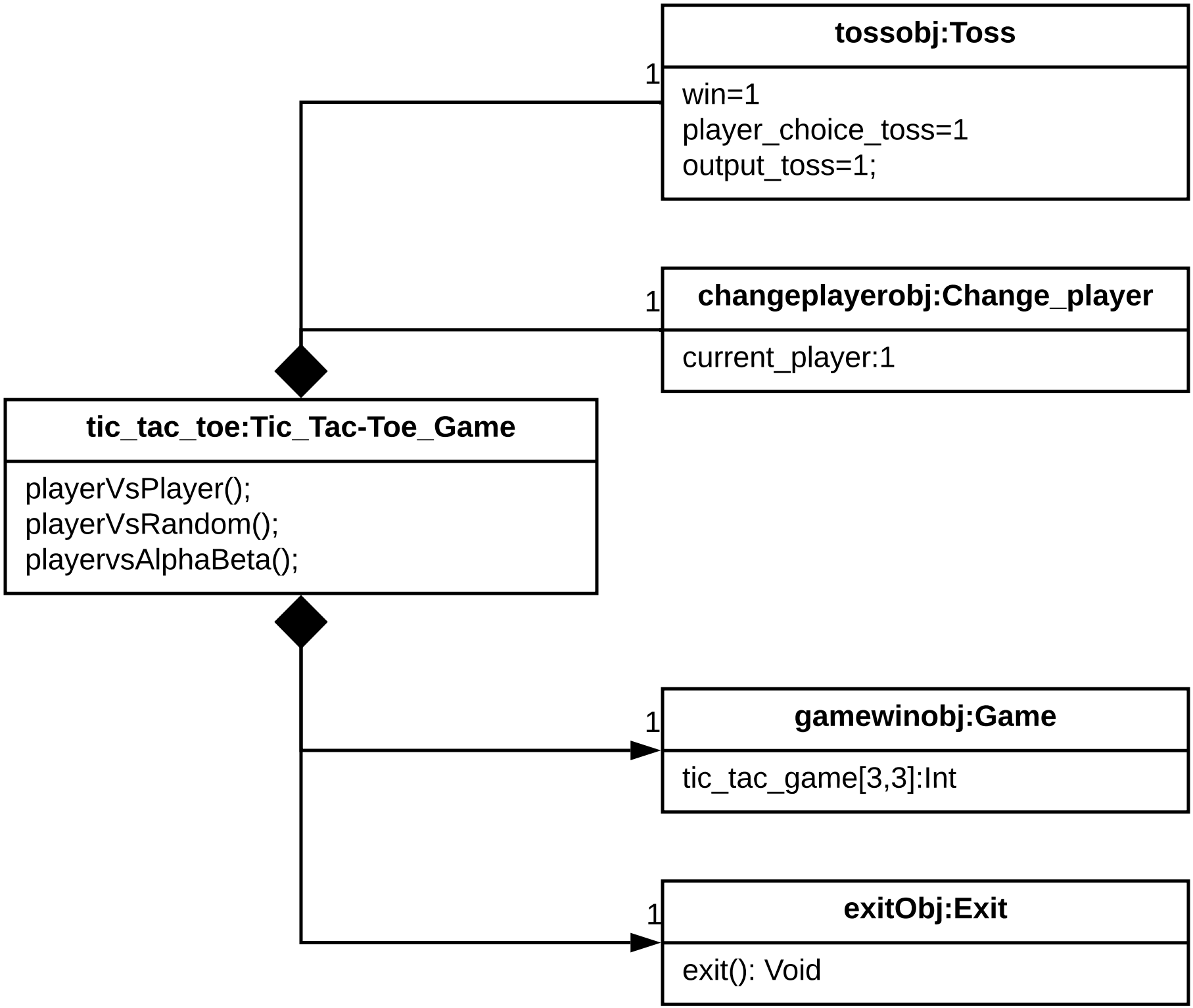
What changes are made from previous versions and Why?

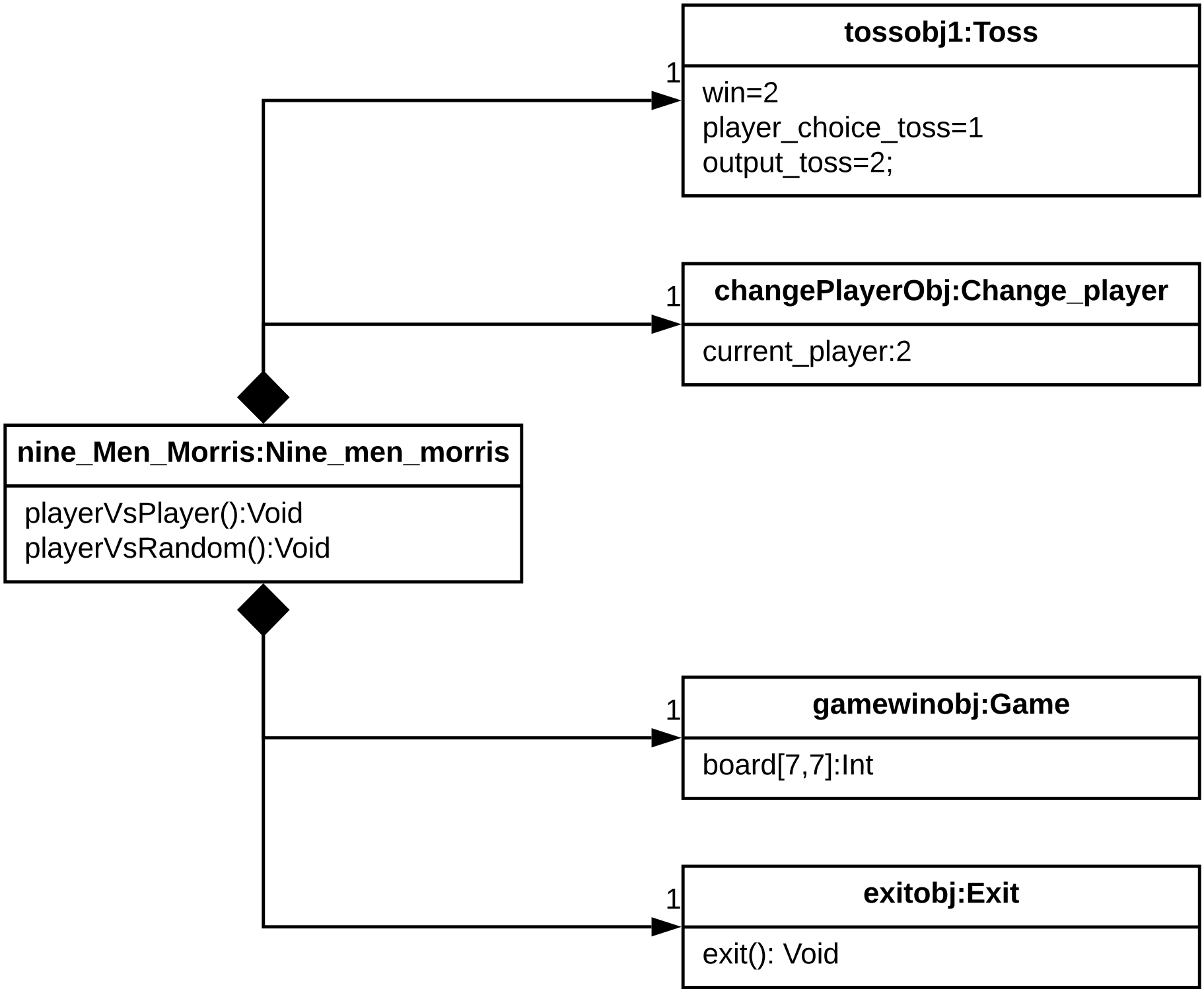
There are 2 major changes from the previous design that was submitted earlier and these changes are the significant improvement in the design.

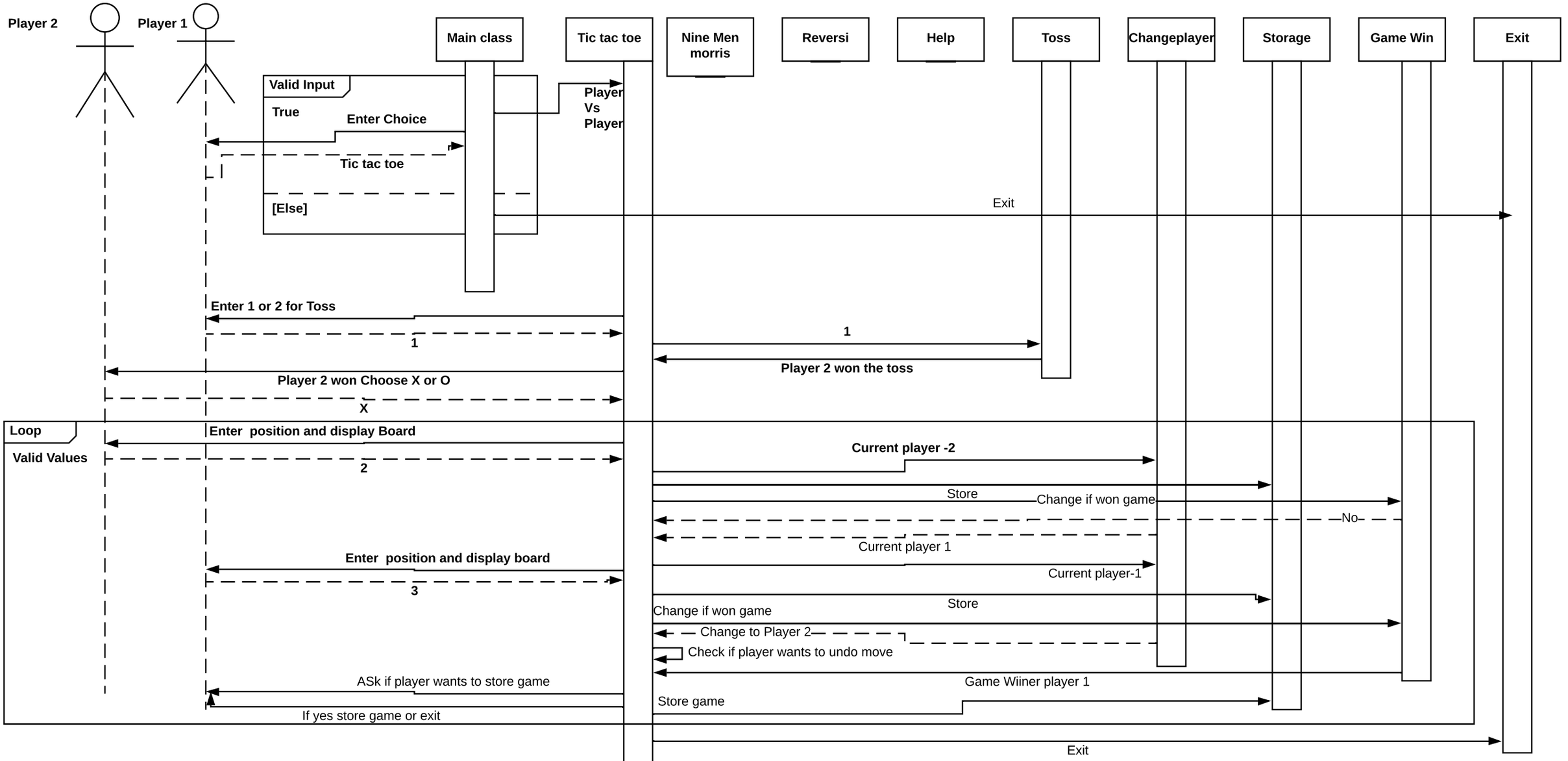
The first significant change is the clubbing of the Random and Alpha-beta algorithm in the respective class. The core reason the change is to reduce the complexity of the program. If these classes were separate they would have the capacity to improve at any given point which is very much valid but in addition to this advantage comes the drawback of memory utilisation, increase in the parameter complexity as the values are passed by value since the reference would change the entire pattern and if the user wants to redo a move there have to be two classes that have to be called every time the user redoes the move. The simple and the reduced complex structure allowed a single memory allocation that is common throughout the class and the changes are made to that variable although the storage is on a different class.

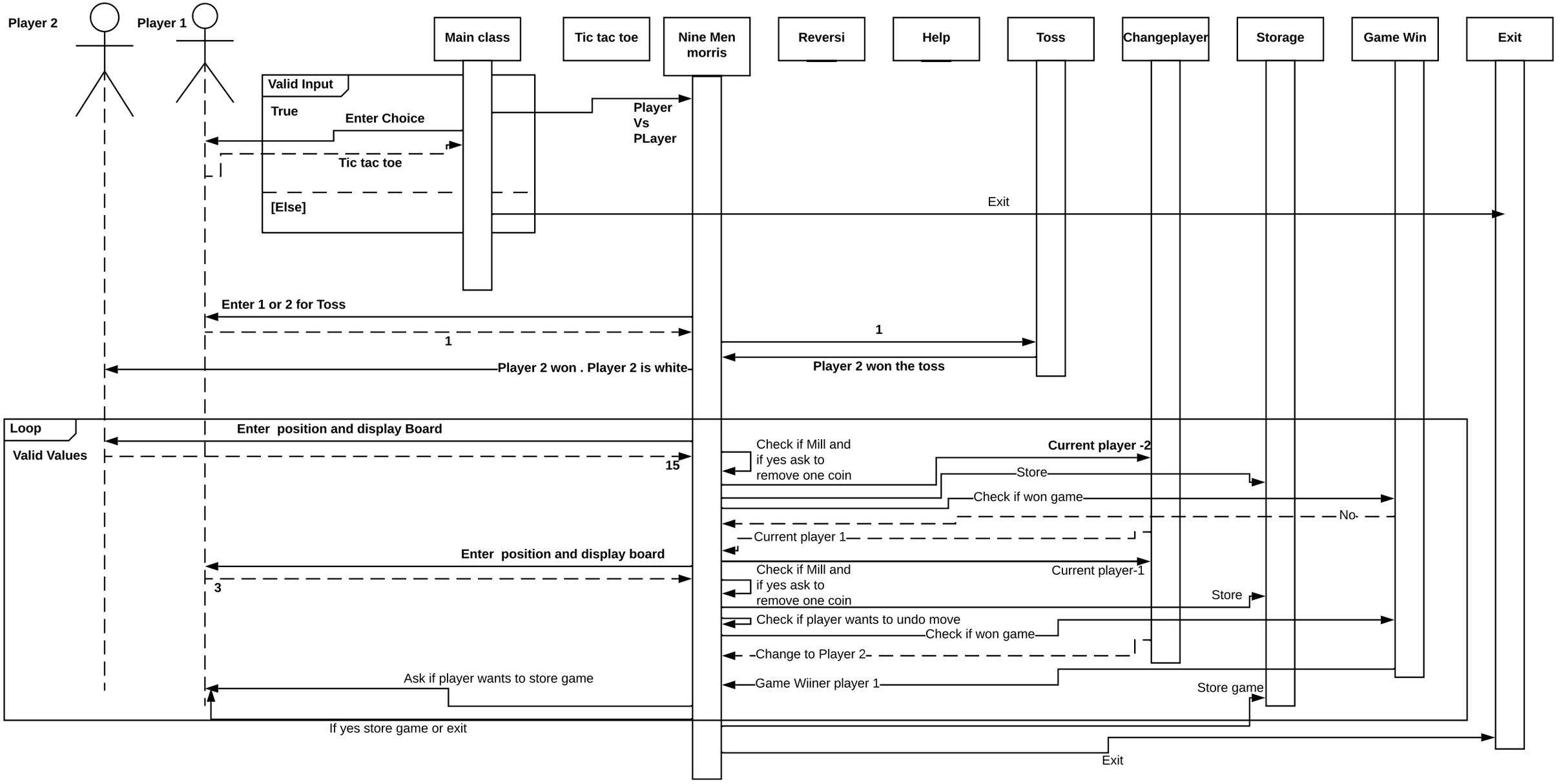
The second significant change is the removal of separate specific function related classes like a board, piece, move and validity into the game-specific class but also adding classes like Toss, Change player, GameWin. The reason for the first act is also similar to the above change. Reducing the complexity, increasing efficiency has always prominent for any design. The function-specific classes were forced to have memory allocation that is common throughout the game. Example, if there were classes like Board that displays the values, it must have the array(The game is built on Array Integer Datatype) values, the move class should have the exact values as it alters all the changes, and the validity class must have the values as in games like Nine Men Morris you should know the point where the removal of the coin is valid and where not possible. The storage would have been all messed up and the complexity would have increased. One of the other reasons is also too much power in classes as if one of the classes by any chance if lost or fails to perform would result in all the games going down whereas the basic small function classes like Toss, Change player can also be redone easily. This opinion might be controversial to some as there might be increases weightage on game-specific classes but the argument would be the same as the explanation that even if one class goes down only one game fails to function but the function-specific class would stop all the games.


Main Program
-gameChoice: Int
-playerChoice: Int
-exit: Bool
+main(): Void
1
Help
+help(): Void
1
Tic_Tac_Toe_Game
-tic_tac_toe: Int [3,3]
-current_player:Int
-player1_value:Int
-player2_value:Int
-position: Int[9]
-enterposition:Int
-random_position:Int
+playervsPlayer():Void
-display(int[,] ,int):Void
-playerVsPlayer_Moves():Void
-input_Validity(int):Bool
-win():Void
-arrayValue():Void
+playerVsRandom():Void
-playervsRandom_moves():Void
-random():Int
1
Nine_men_morris
-board: Int[7,7]
-random_position:Int
-black:Int
-white:Int
-white_piece_counter:Int
-black_piece_counter:Int
-player_1:Int
-player_2:Int
-playermill:Int
-remove_position:Int
-current_position:Int
-position:Int[24]
-enter_position:Int
-board_moves:Int[7,7]
-counter1 to counter16:Int
+playerVsPlayer():Void
-display(int [,] , int):Void
-moves():Void
-array_Value:Void
-mill():Bool
-remove_piece():Void
+playerVsRandom_moves():Void
-random:Int
1
Reversi
-black:Int
-white:Int
-white_piece_counter:Int
-black_piece_counter:Int
-player_1:Int
-player_2:Int
-position:Int[]
-board_moves:Int[]
+playerVsPlayer():Void
-display(int[,] , int):Void
-moves(): Void
-arrayValue():Void
-reversi(): Int
+playerVsRandom_moves();Void
-random:Int
1
Toss
-win:Int
-toss:Int
-outposs:Int
-toss():Int
1
Change_player
-current_player
+changeplayer(int): Int
Exit
exit(): Void
Game_win
- tic_tac_toe:Int
+game_win(int [,]): Bool
Overlap
-enterposition:Int
-position[]:Int
+overlap_tic_tac_toe(int, int[]): Bool
+overlap_nine_men_morris(int, int[]): Bool


tossobj:Toss
1
win=1
player_choice_toss=1
output_toss=1;
changeplayerobj:Change_player
1
current_player:1
tic_tac_toe:Tic_Tac-Toe_Game
playerVsPlayer();
playerVsRandom();
playervsAlphaBeta();
gamewinobj:Game
1
tic_tac_game[3,3]:Int
exitObj:Exit
1
exit(): Void


tossobj1:Toss
1
win=2
player_choice_toss=1
output_toss=2;
changePlayerObj:Change_player
1
current_player:2
nine_Men_Morris:Nine_men_morris
playerVsPlayer():Void
playerVsRandom():Void
gamewinobj:Game
1
board[7,7]:Int
exitobj:Exit
1
exit(): Void


Player 2
Player 1
Main class
Tic tac toe
Nine Men morris
Reversi
Help
Toss
Changeplayer
Storage
Game Win
Exit
Valid Input
True
Enter Choice
Tic tac toe
[Else]
Player Vs Player
Exit
Enter 1 or 2 for Toss
1
1
Player 2 won the toss
Player 2 won Choose X or O
X
Loop
Valid Values
Enter position and display Board
2
Current player -2
Store
Change if won game
No
Current player 1
Enter position and display board
3
Current player-1
Store
Change if won game
Change to Player 2
Check if player wants to undo move
Game Wiiner player 1
ASk if player wants to store game
If yes store game or exit
Store game
Exit


Player 2
Player 1
Main class
Tic tac toe
Nine Men morris
Reversi
Help
Toss
Changeplayer
Storage
Game Win
Exit
Valid Input
True
Enter Choice
Tic tac toe
[Else]
Player Vs PLayer
Exit
Enter 1 or 2 for Toss
1
1
Player 2 won the toss
Player 2 won . Player 2 is white
Loop
Valid Values
Enter position and display Board
15
Check if Mill and if yes ask to remove one coin
Current player -2
Store
Check if won game
No
Current player 1
Enter position and display board
3
Current player-1
Check if Mill and if yes ask to remove one coin
Store
Check if player wants to undo move
Check if won game
Change to Player 2
Ask if player wants to store game
Game Wiiner player 1
Store game
If yes store game or exit
Exit

How can the Program be executed?

The program is a very simple C# code that is the name of Oopsjava, strange name, have no reason for that name. Found it cute might change it later. The first set of instructions would give you options for choice of game and the choice of player. Then you would be displayed the game board with the position number and then is the Toss time. One of the two-player wins the toss and then chooses the piece that he wants to play with. Toss is specific only to Player vs Player and not included in AI. After passing validity moves you console displays the board at every point and asks for moves. The winner is stated at the end. The same thing happens for the AI expect for the toss and also the algorithm is Random so it might be very stupid and easy to beat. Further enhancements in Alpha-beta algorithm can be done and scope and design have been routed for that strategy. The help system would just go to links and even here further scope in terms of Hints feature and better Help commands are routed in design but not implemented.

Design pattern

The design pattern that I think Is the most suitable is a composite pattern for this design. The composite pattern can be identified by the fact that there is a single parent class and then it boils down into three separate classes going according to the respective system according to the choice that is chosen by the user. One of the major requirements of the project is to be extensible and hence compositive pattern is used as it is very easy with the pattern that can extend. The parent class Is the main class and it is easier with the composite pattern as it allows the game to go down with a choice that is required to split into three different games namely Tic tac toe, Reversi and Nine Men Morris. The game then again shifts into a model where there Is only one class and the rest of the instances from different functional type classes are combined Into one single class which is the main game-specific class.

The software pattern is done implemented with SOLID, DRY, KISS, YAGNI patterns. The single function specific classes like the ChangePlayer class, GameWin are done according to the SOLID pattern where the classes don't hold too much of the power and have only instruction specific information on them. The contradictory of which some critics might think of giving too much power on the specific game classes holds has been given a reason in the section why the changes in the design from the previous design earlier in the report. The DRY pattern is a very strict pattern that Is used where all the attributes declared in the system have been only that are of absolute importance. The reason for more function-related variables is the same, less scope less power and not much repeatability. KISS, the way this principle is implemented in the system is through the functions. The functions in the class have a very straight objective and have been given not more than one or two tasks to perform so that commands are very easily readable and can be easily debugged in Future. YAGNI software implementation can be seen through the implementation of specific Random function instead of creating a class as there might be a huge difference in the time this code takes to deliver and memory allocation if they are put into a separate class.

Reusable Class

The reusable classes are more of function Specific and not game-specific classes in this design pattern. The classes are ChangePlayer, GameWin, Storage, Exit and Toss that can be implemented anywhere with the basic arguments that are needed.