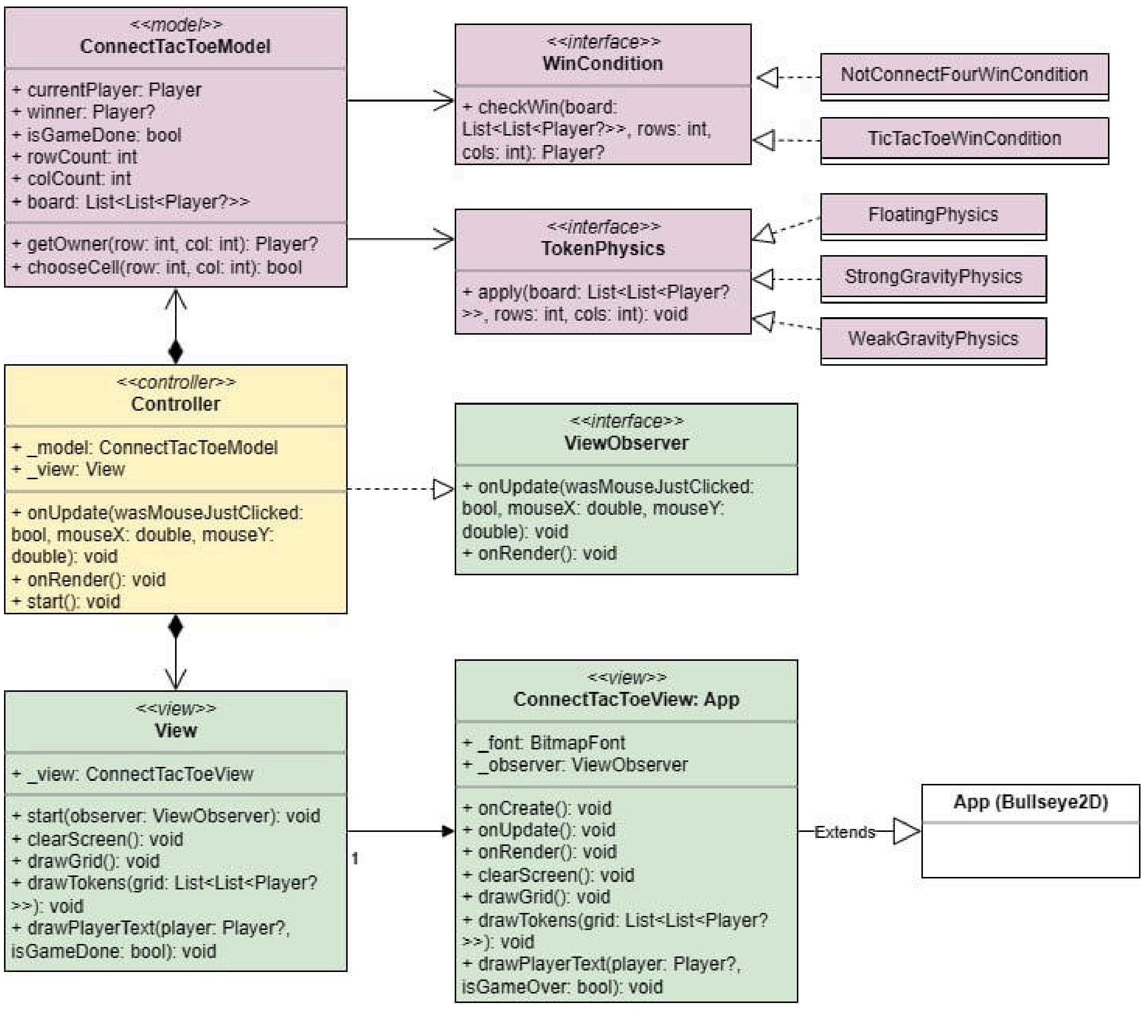
Lab Report 3
CS 150

Dale Sealtiel T. Flores
2023-11373

Maricris Mandario
2022-02970

1. Provide a class diagram containing all the classes in your submission using either UML or the informal UML-like notation used in class (ensuring that interfaces are distinguishable from concrete classes). Additionally, provide a visual indicator of which classes belong to the model, view, controller.
   Include the Bullseye2d `App` class. Include all public fields and methods except for those of the Bullseye2D `App` class.


<<model>>
ConnectTacToeModel
+ currentPlayer: Player
+ winner: Player?
+ isGameDone: bool
+ rowCount: int
+ colCount: int
+ board: List<List<Player?>>
+ getOwner(row: int, col: int): Player?
+ chooseCell(row: int, col: int): bool
<<interface>>
WinCondition
+ checkWin(board: List<List<Player?>>, rows: int, cols: int): Player?
NotConnectFourWinCondition
TicTacToeWinCondition
<<interface>>
TokenPhysics
+ apply(board: List<List<Player?>>, rows: int, cols: int): void
FloatingPhysics
StrongGravityPhysics
WeakGravityPhysics
<<controller>>
Controller
+ _model: ConnectTacToeModel
+ _view: View
+ onUpdate(wasMouseJustClicked: bool, mouseX: double, mouseY: double): void
+ onRender(): void
+ start(): void
<<interface>>
ViewObserver
+ onUpdate(wasMouseJustClicked: bool, mouseX: double, mouseY: double): void
+ onRender(): void
<<view>>
View
+ _view: ConnectTacToeView
+ start(observer: ViewObserver): void
+ clearScreen(): void
+ drawGrid(): void
+ drawTokens(grid: List<List<Player?>>): void
+ drawPlayerText(player: Player?, isGameDone: bool): void
1
<<view>>
ConnectTacToeView: App
+ _font: BitmapFont
+ _observer: ViewObserver
+ onCreate(): void
+ onUpdate(): void
+ onRender(): void
+ clearScreen(): void
+ drawGrid(): void
+ drawTokens(grid: List<List<Player?>>): void
+ drawPlayerText(player: Player?, isGameOver: bool): void
Extends
App (Bullseye2D)

2. Show how your code follows the *Open-Closed Principle* **using composition** for the following scenarios by providing working code snippets that implement them and showing how `ConnectTacToeModel` should be instantiated to accomodate them *(do* ***not*** *add them to your* `.dart` *files; no need to maximize code reuse; no need to change* `WinConditionType`*,* `TokenPhysicsType`*, and* `make`*)*:

   (a) A new **Connect Four** win condition variant is introduced in which the first player who has four or more tokens in a contiguous configuration horizontally, vertically, or diagonally wins.

```
others.dart
// existing code
class ConnectFourWinCondition implements WinCondition {
  @override
  (Player?, Player?) checkWin(List<List<Player?>> board, int rows, int cols) {
    bool p1Wins = false;
    bool p2Wins = false;

    bool dfs(int row, int col, int dRow, int dCol, Player player, int count) {
      if (count == 4) return true;
      int r = row + dRow;
      int c = col + dCol;
      if (r < 0 || r >= rows || c < 0 || c >= cols) return false;
      if (board[r][c] != player) return false;
      return dfs(r, c, dRow, dCol, player, count + 1);
    }

    for (int row = 0; row < rows; row++) {
      for (int col = 0; col < cols; col++) {
        Player? player = board[row][col];
        if (player == null) continue;
        List<List<int>> directions = [
          [0, 1],
          [1, 0],
          [1, 1],
          [1, -1],
        ];
        for (var dir in directions) {
          if (dfs(row, col, dir[0], dir[1], player, 1)) {
            if (player == Player.p1) p1Wins = true;
            if (player == Player.p2) p2Wins = true;
          }
        }
      }
    }

    if (p1Wins && p2Wins) return (Player.p1, Player.p2);
    if (p1Wins) return (Player.p1, null);
    if (p2Wins) return (Player.p2, null);
    return (null, null);
  }
}
//existing code

main.dart
void main(){
   //existing code
   switch (params['win']){
   // existing code
      case 'connectfour':
         winConditionType = WinConditionType.connectfour;
         break;
   }

      //existing code

   switch (winConditionType) {
      //existing code
      case WinConditionType.connectfour:
         winCondition = ConnectFourWinCondition();
   }

   // existing code

   if (winCondition != null && tokenPhysics != null) {
    ConnectTacToeModel model = ConnectTacToeModel(
      winCondition: winCondition,
      tokenPhysics: tokenPhysics,
    );
    View view = View();
    Controller controller = Controller(model, view);

    controller.start();
  }
}
```

(b) A new **Two Sides** token physics variant is introduced that has tokens move towards the topmost row if places in the first three rows, or the bottommost row if placed in the last three rows *(i.e., same effect as **Strong Gravity** for the last three rows)*

others.dart

```
// existing code
class TwoSidesPhysics implements TokenPhysics {
  @override
  void apply(List<List<Player?>> board, int rows, int cols) {
    for (int col = 0; col < cols; col++) {
      for (int row = rows - 1; row >= 3; row--) {
        if (board[row][col] == null) {
          for (int below = row - 1; below >= 3; below--) {
            if (board[below][col] != null) {
              board[row][col] = board[below][col];
              board[below][col] = null;
              break;
            }
          }
        }
      }
    }
    for (int col = 0; col < cols; col++) {
      for (int row = 0; row <= 2; row++) {
        if (board[row][col] == null) {
          for (int above = row + 1; above <= 2; above++) {
            if (board[above][col] != null) {
              board[row][col] = board[above][col];
              board[above][col] = null;
              break;
            }
          }
        }
      }
    }
  }
}
//existing code
```

main.dart

```
void main(){
   //existing code

   switch(params['physics']) {
      //existing code
      case 'twoSides':
         tokenPhysicsType = TokenPhysicsType.twoSides;
   }

   //existing code
   switch (tokenPhysicsType) {
      //existing code
      case TokenPhysicsType.twoSides:
         tokenPhysics = TwoSidesPhysics();
   }

   if (winCondition !+ null && tokenPhysics != null) {
      ConnectTacToeModel model = ConnectTacToeModel(
         winCondition: winCondition,
         tokenPhysics: tokenPhysics,
      );
      View view = View();
      Controller controller = Controller(model, view);
      controller.start
   }
}
```