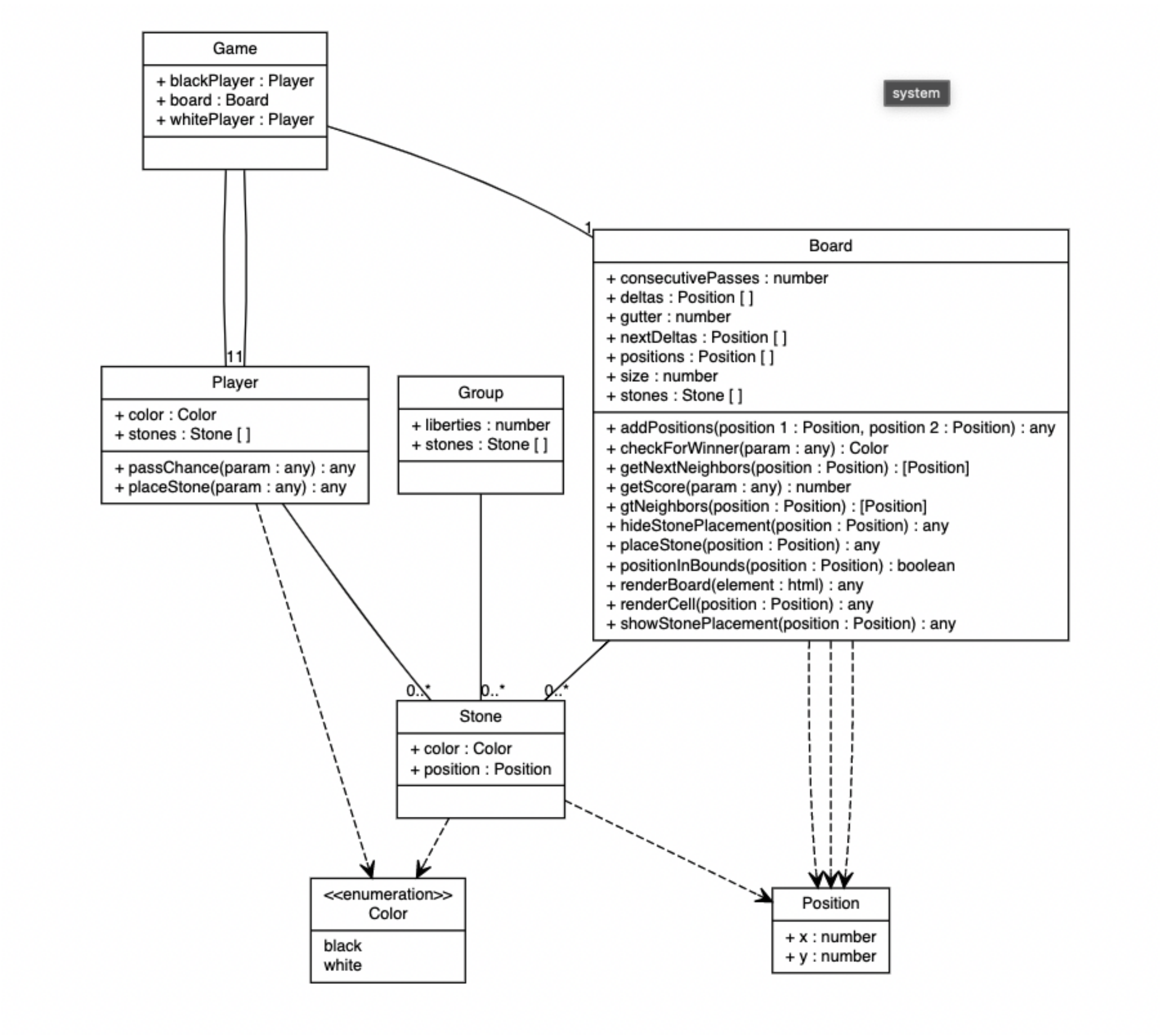
1. Names
    - Nienfu Hsieh
    - Drew Dugan

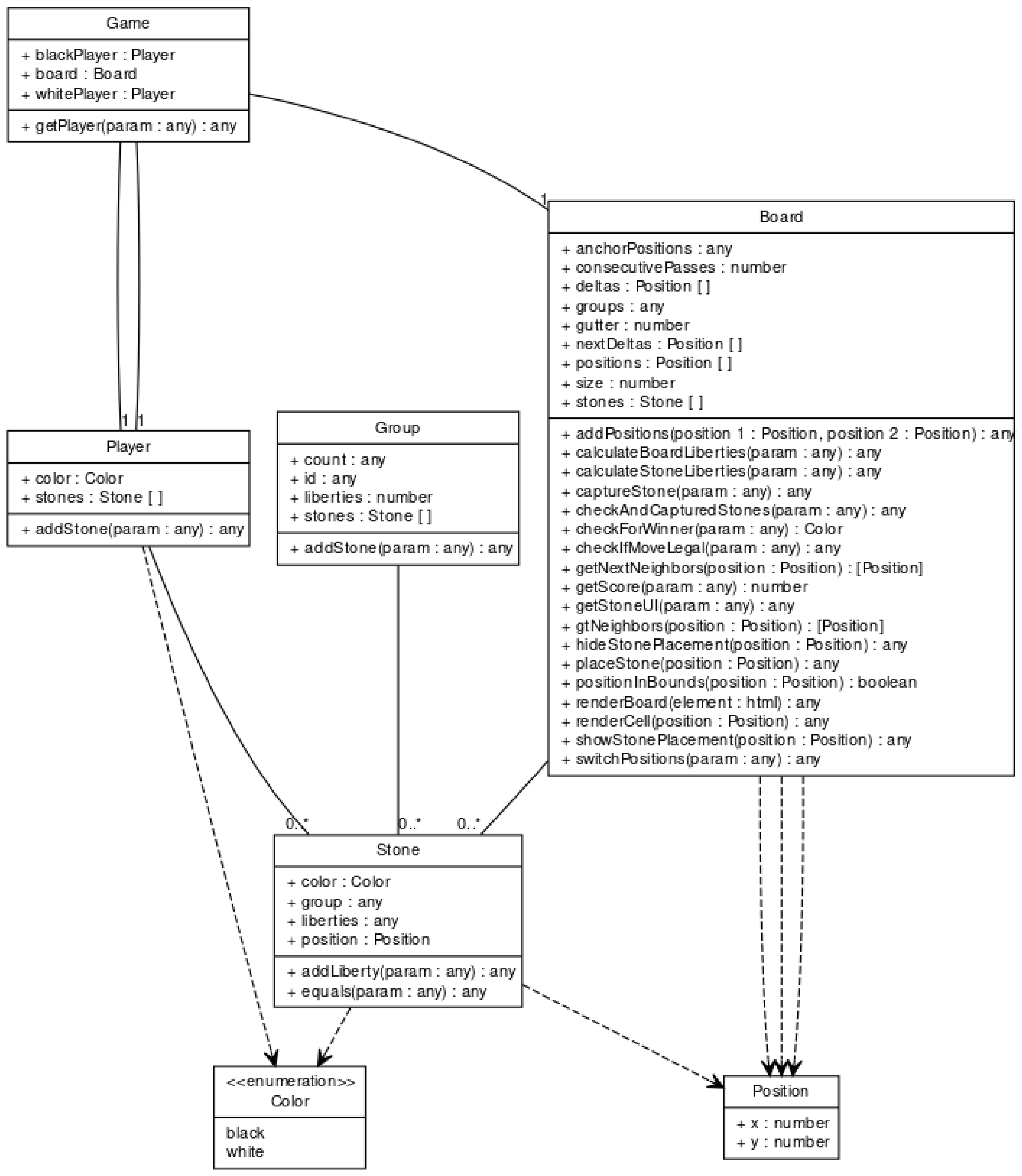
2. Final state of the system:
    - In this Iteration, the game logic was updated to be more robust and covers all the rules and exceptions of the game GO. Specifically, the positional superko rule and suicide-move exception were implemented. A ‘Game Over’ condition was added as well. Now any two players can play this game the same as in real life. ‘Undo’ functionality and ‘Game history’ features were not implemented due to time constraints.

3. UML diagrams
   Project 5:

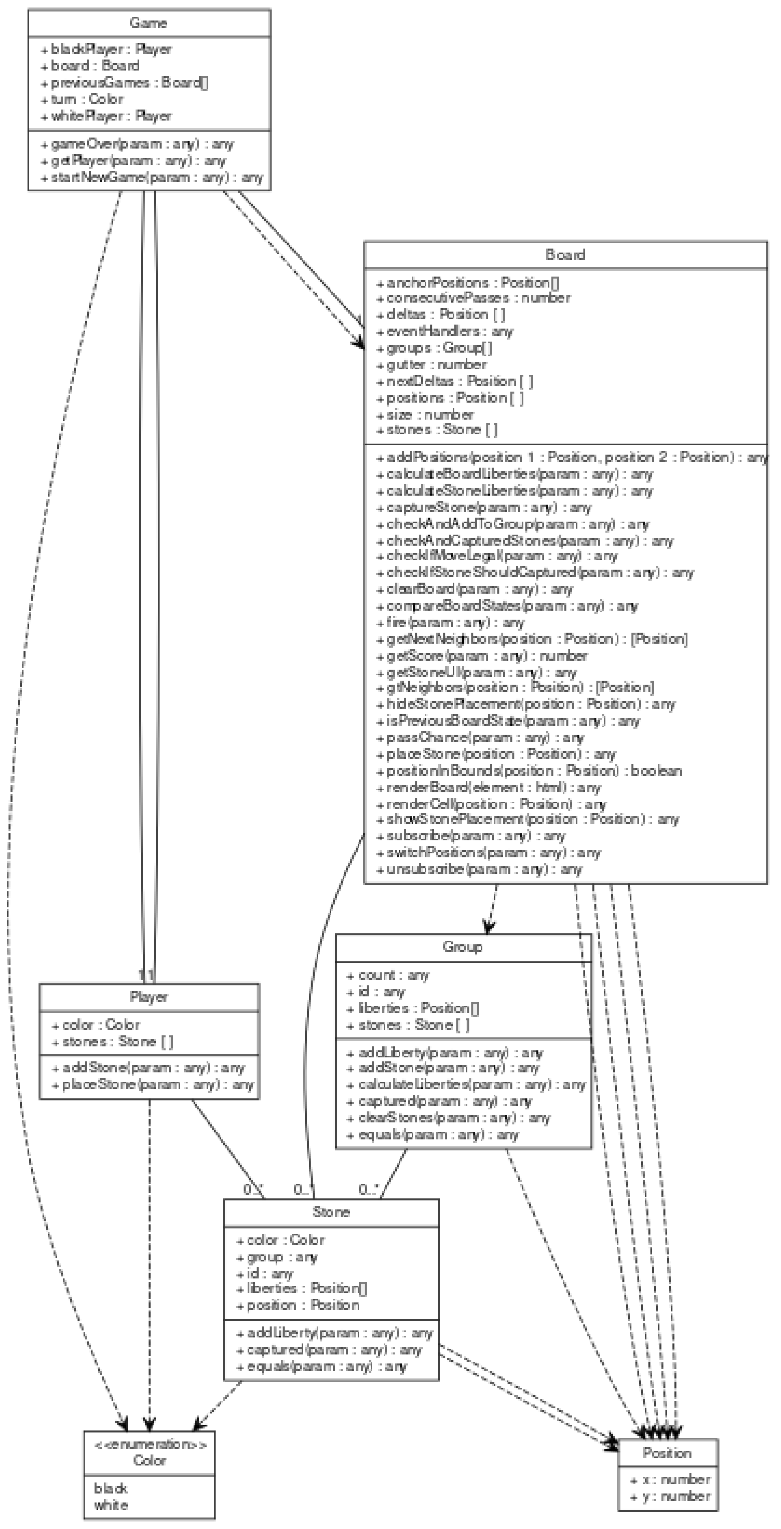


Game
+ blackPlayer : Player
+ board : Board
+ whitePlayer : Player
system
1
11
Board
+ consecutivePasses : number
+ deltas : Position [ ]
+ gutter : number
+ nextDeltas : Position [ ]
+ positions : Position [ ]
+ size : number
+ stones : Stone [ ]
+ addPositions(position 1 : Position, position 2 : Position) : any
+ checkForWinner(param : any) : Color
+ getNextNeighbors(position : Position) : [Position]
+ getScore(param : any) : number
+ gtNeighbors(position : Position) : [Position]
+ hideStonePlacement(position : Position) : any
+ placeStone(position : Position) : any
+ positionInBounds(position : Position) : boolean
+ renderBoard(element : html) : any
+ renderCell(position : Position) : any
+ showStonePlacement(position : Position) : any
Player
+ color : Color
+ stones : Stone [ ]
+ passChance(param : any) : any
+ placeStone(param : any) : any
Group
+ liberties : number
+ stones : Stone [ ]
0..*
0..*
0..*
Stone
+ color : Color
+ position : Position
<<enumeration>>
Color
black
white
Position
+ x : number
+ y : number

Project 6:


Game
+ blackPlayer : Player
+ board : Board
+ whitePlayer : Player
+ getPlayer(param : any) : any
1
Board
+ anchorPositions : any
+ consecutivePasses : number
+ deltas : Position [ ]
+ groups : any
+ gutter : number
+ nextDeltas : Position [ ]
+ positions : Position [ ]
+ size : number
+ stones : Stone [ ]
+ addPositions(position 1 : Position, position 2 : Position) : any
+ calculateBoardLiberties(param : any) : any
+ calculateStoneLiberties(param : any) : any
+ captureStone(param : any) : any
+ checkAndCapturedStones(param : any) : any
+ checkForWinner(param : any) : Color
+ checkIfMoveLegal(param : any) : any
+ getNextNeighbors(position : Position) : [Position]
+ getScore(param : any) : number
+ getStoneUI(param : any) : any
+ gtNeighbors(position : Position) : [Position]
+ hideStonePlacement(position : Position) : any
+ placeStone(position : Position) : any
+ positionInBounds(position : Position) : boolean
+ renderBoard(element : html) : any
+ renderCell(position : Position) : any
+ showStonePlacement(position : Position) : any
+ switchPositions(param : any) : any
1 1
Player
+ color : Color
+ stones : Stone [ ]
+ addStone(param : any) : any
Group
+ count : any
+ id : any
+ liberties : number
+ stones : Stone [ ]
+ addStone(param : any) : any
0..*
0..*
0..*
Stone
+ color : Color
+ group : any
+ liberties : any
+ position : Position
+ addLiberty(param : any) : any
+ equals(param : any) : any
<<enumeration>>
Color
black
white
Position
+ x : number
+ y : number

Final UML diagram


Game
+ blackPlayer : Player
+ board : Board
+ previousGames : Board[]
+ turn : Color
+ whitePlayer : Player
+ gameOver(param : any) : any
+ getPlayer(param : any) : any
+ startNewGame(param : any) : any
Board
+ anchorPositions : Position[]
+ consecutivePasses : number
+ deltas : Position [ ]
+ eventHandlers : any
+ groups : Group[]
+ gutter : number
+ nextDeltas : Position [ ]
+ positions : Position [ ]
+ size : number
+ stones : Stone [ ]
+ addPositions(position 1 : Position, position 2 : Position) : any
+ calculateBoardLiberties(param : any) : any
+ calculateStoneLiberties(param : any) : any
+ captureStone(param : any) : any
+ checkAndAddToGroup(param : any) : any
+ checkAndCapturedStones(param : any) : any
+ checkIfMoveLegal(param : any) : any
+ checkIfStoneShouldCaptured(param : any) : any
+ clearBoard(param : any) : any
+ compareBoardStates(param : any) : any
+ fire(param : any) : any
+ getNextNeighbors(position : Position) : [Position]
+ getScore(param : any) : number
+ getStoneUI(param : any) : any
+ gtNeighbors(position : Position) : [Position]
+ hideStonePlacement(position : Position) : any
+ isPreviousBoardState(param : any) : any
+ passChance(param : any) : any
+ placeStone(position : Position) : any
+ positionInBounds(position : Position) : boolean
+ renderBoard(element : html) : any
+ renderCell(position : Position) : any
+ showStonePlacement(position : Position) : any
+ subscribe(param : any) : any
+ switchPositions(param : any) : any
+ unsubscribe(param : any) : any
1
1 1
Player
+ color : Color
+ stones : Stone [ ]
+ addStone(param : any) : any
+ placeStone(param : any) : any
Group
+ count : any
+ id : any
+ liberties : Position[]
+ stones : Stone [ ]
+ addLiberty(param : any) : any
+ addStone(param : any) : any
+ calculateLiberties(param : any) : any
+ captured(param : any) : any
+ clearStones(param : any) : any
+ equals(param : any) : any
0..*
0..*
0..*
Stone
+ color : Color
+ group : any
+ id : any
+ liberties : Position[]
+ position : Position
+ addLiberty(param : any) : any
+ captured(param : any) : any
+ equals(param : any) : any
<<enumeration>>
Color
black
white
Position
+ x : number
+ y : number

OOAD patterns:

- The Observer pattern was used in the **Board** class.
    1. The standard subscribe, unsubscribe and fire methods were defined but instead of just having one array for handlers, we implemented an improved event subscription system that can fire different event handlers depending on the event fired.
- **Game** class subscribes to two events on the board object.
    1. playerTurn event - fired when a player successfully places a stone on the board or passes their chance.
    2. gameOver event - fired when both the players pass their chances in succession.

Changes since project 6:

- Observer pattern implemented
- Game over mechanism added
- Logic to determine positional superko added
- Logic to determine whether a suicide move is legal
- Grouping mechanism improved
- Game:
    - "turn": "property",
    - "previousGames": "property",
    - "getPlayer": "method",
    - "startNewGame": "method",
    - "gameOver": "method"
- Player
    - "addStone": "method",
    - "placeStone": "method"
- Stone
    - "captured":"method"
- Group
    - "calculateLiberties": "method",
    - "captured": "method",
    - "equals": "method",
    - "clearStones": "method"
- Board
    - "eventHandlers": "property",
    - "subscribe": "method",
    - "unsubscribe": "method",
    - "fire": "method",
    - "isPreviousBoardState": "method",

    - "compareBoardStates": "method",
    - "checkIfStoneShouldCaptured": "method",
    - "checkAndAddToGroup": "method",
    - "passChance": "method",
    - "clearBoard": "method"

4. Third party code:
    - Vite was used as a build tool.
    - Lodash was used for Array/Object utilities.
    - Recycled and improved Observer pattern code from this article.
    - Inspiration for some of the **Board** class methods was drawn from Weiqi

5. Statement on OOAD:
    - Learning the usage of observer pattern was quite the eye-opener in regards to the internal workings of an event-driven architecture.
    - Dealing with scope and the '*this*' keyword was tricky.
    - Iterating over the classes/methods/properties and refining them to make it