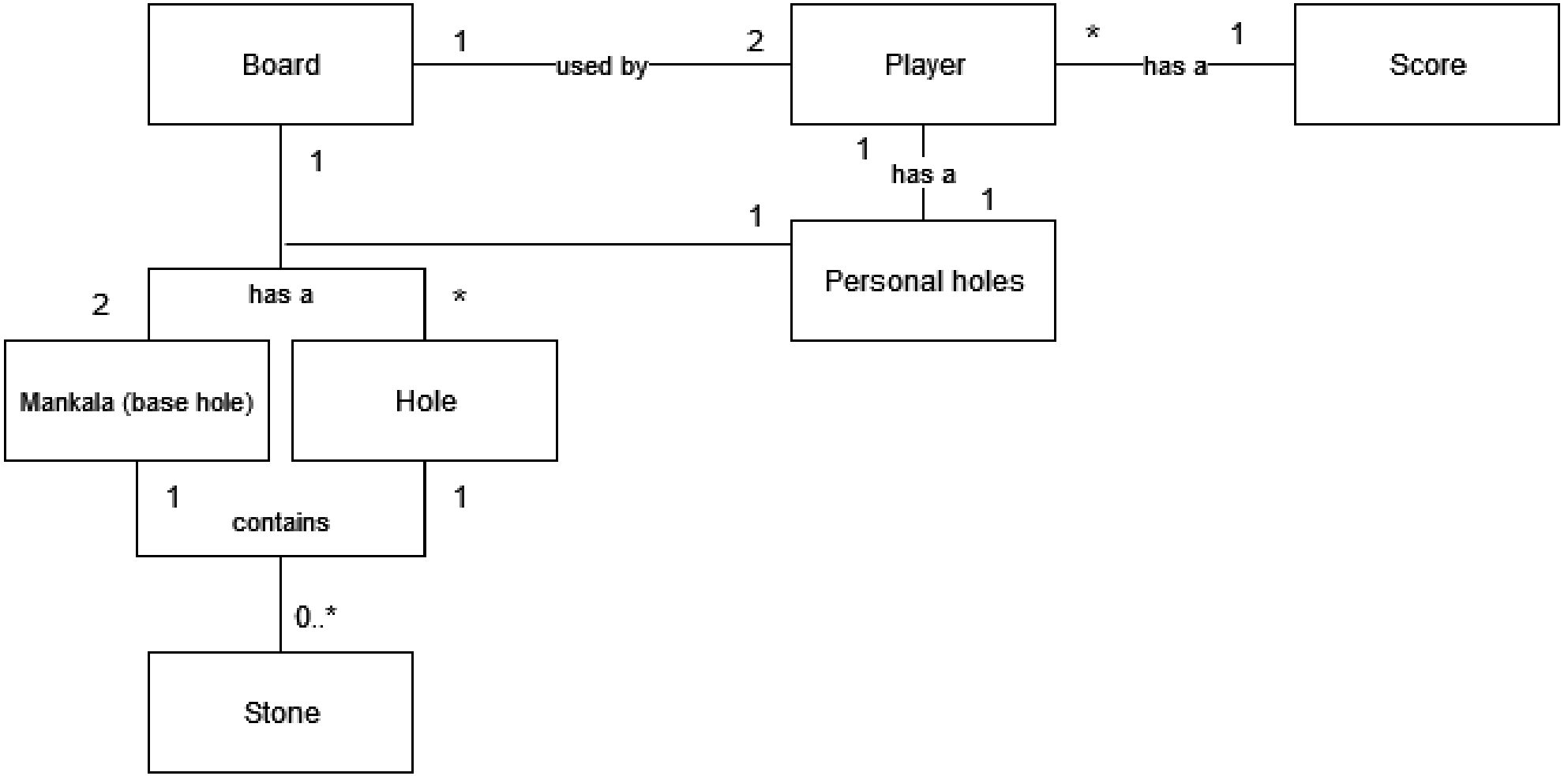
Domain model



CVA scheme

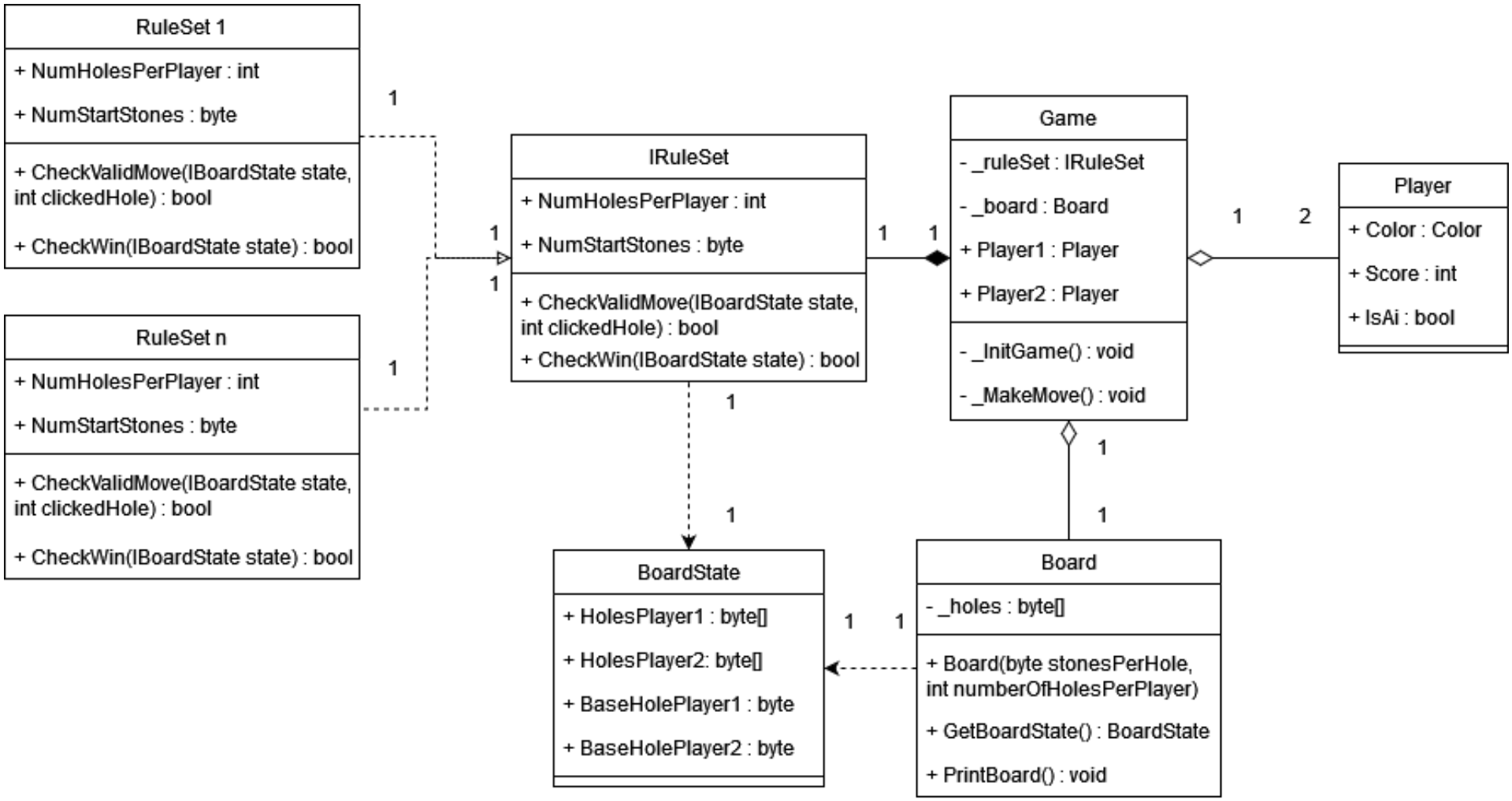
Commonality	Variations
Board	8 holes, 6 holes, n holes
Stone	White stones, black holes
Starting N.O. stones	4, 6, n
Rules for win	Ruleset 1, 2 3
Rules for move	Ruleset 1, 2 3

Analysis matrix

Use-case	Ruleset 1	Ruleset 2	Ruleset 3
Score is calculated	Score equals to number of stones in Basehole		
Determine winner	Player with highest score wins		
Perform next move	Spread the stones in selected hole over the next (CCW) holes, where every hole receives one stone		

The number of stones, holes and color of said stones are not determined by the ruleset but by the player and can be changed per game. Thus, it is not included in the analysis matrix.

Class diagram



In our class diagram we have chosen to apply a strategy pattern to IRuleSet, the ruleset of the game. This way we allow the Game class to use RuleSet methods without depending on a specific implementation. This helps by ensuring that the Game class can use RuleSet methods for different sets of rules. We believe this to be the best pattern to use because we want to use different variants of an algorithm within an object and be able to switch from one algorithm to another during runtime. This prevents hard-wiring of all the algorithms into the program.

We use the helper class Boardstate to prevent us from hard coding too much logic. While this is not a design pattern, we do feel the need to explain this choice. This method helps reduce code coupling by taking stress off the link between IRuleset and board. This way we keep our code neat, efficient and future proof.

We chose for a strategy and not for a bridge pattern because we dont need to decouple an abstraction from its implementation here.

- Game
    - This class is responsible for storing the board, the rules, 2 players, starting a new game, and making new moves and applying other game logic
    - It meets these responsibilities by using the methods _InitGame() and _MakeMove(). The game loop is maintained by _MakeMove(), and it ends when _ruleSet.CheckWin() returns true.
- Player
    - This class is responsible for storing the player values color, score, and whether it is an AI or not
    - This class fulfills its requirements just storing the info :)
- Board
    - This class is responsible for the amount of stones in each hole, getting the boardstate, and printing a representation of the current board to the screen. It takes care of manipulating the board and interpreting it.
    - The board class succesfully creates the promised abstraction layer by supplying a few methods like PrintBoard(), which makes interacting with the board easy with low coupling.
- BoardState

    - This class acts as an abstraction on the board, to represent the current state of the board. This way, the implementation of the board is decoupled from the important information used by e.g. IRuleSet.
    - This meets his responsibilities by just simply storing the given information, which will always meet the expectations of the dependent classes.
- IRuleSet
    - This interface is responsible for the starting values of the amount of holes per player, the amount of starting stones per hole, checking the validity of a move and determining the winner of a game. It houses all logics belonging to a given ruleset.
    - This interface matches his responsibilities by providing many methods that take care of the relevant logic. Because it's an interface, the implementations are abstracted from the usage, as required for the strategy pattern.
- RuleSet1
    - This is an arbitrary implementation of IRuleSet, and acts solely as an example of the strategy pattern.
    - This implementation meets the given responsibilities by actually implementing the IRuleSet interface, filling it with the said logic.
- RuleSet2
    - This is an arbitrary implementation of IRuleSet, and acts solely as an example of the strategy pattern.
    - This implementation meets the given responsibilities by actually implementing the IRuleSet interface, filling it with the said logic.

Evaluation

Our program is easily expandible because of how abstract we have chosen to make it. Since we make use of the strategy pattern, all that is needed to add a new ruleset is another implementation of the IRuleSet interface. This way nothing within the program changes except for the rules.