Project 2 UML Charts

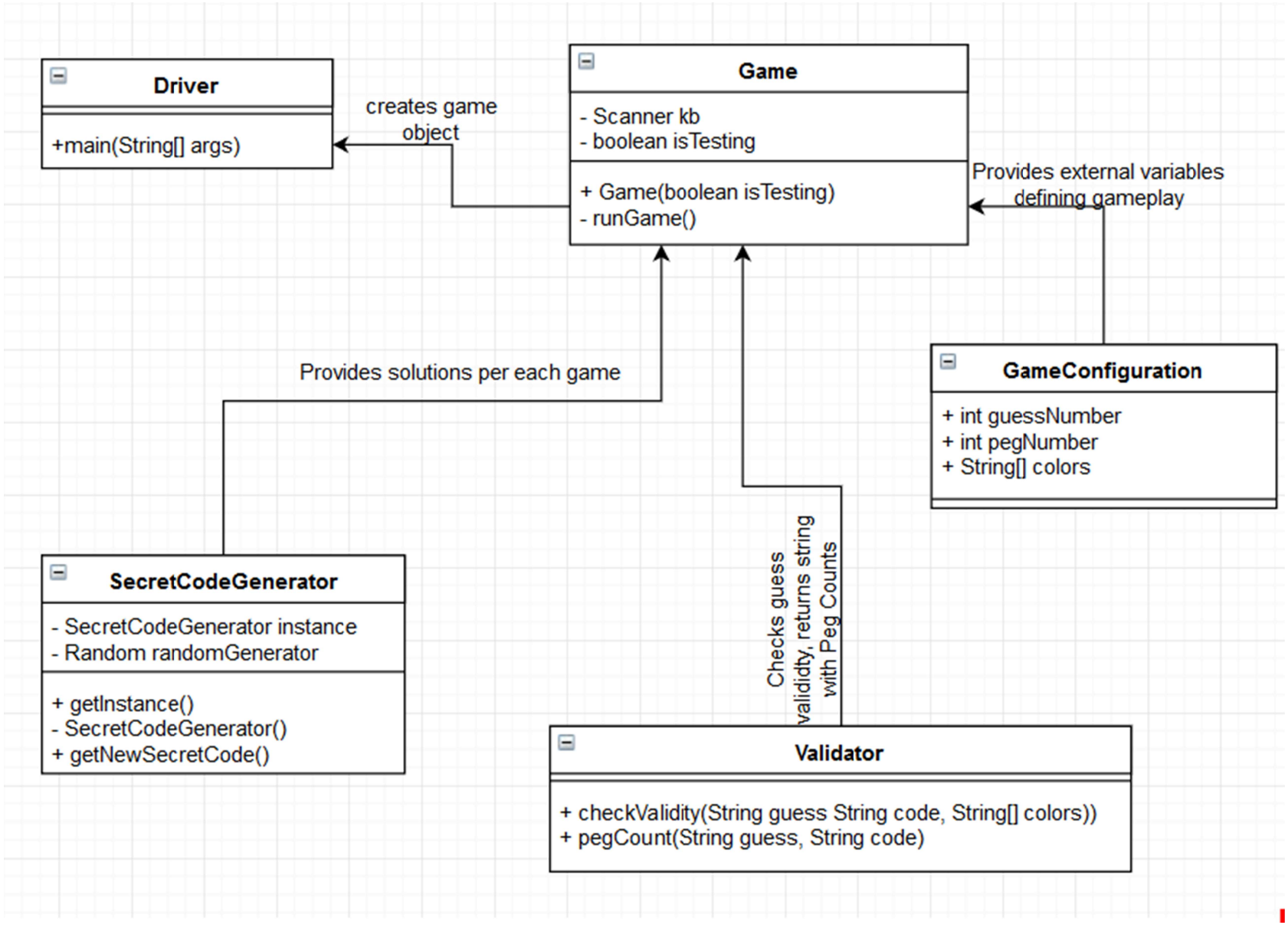
Suhas Raja

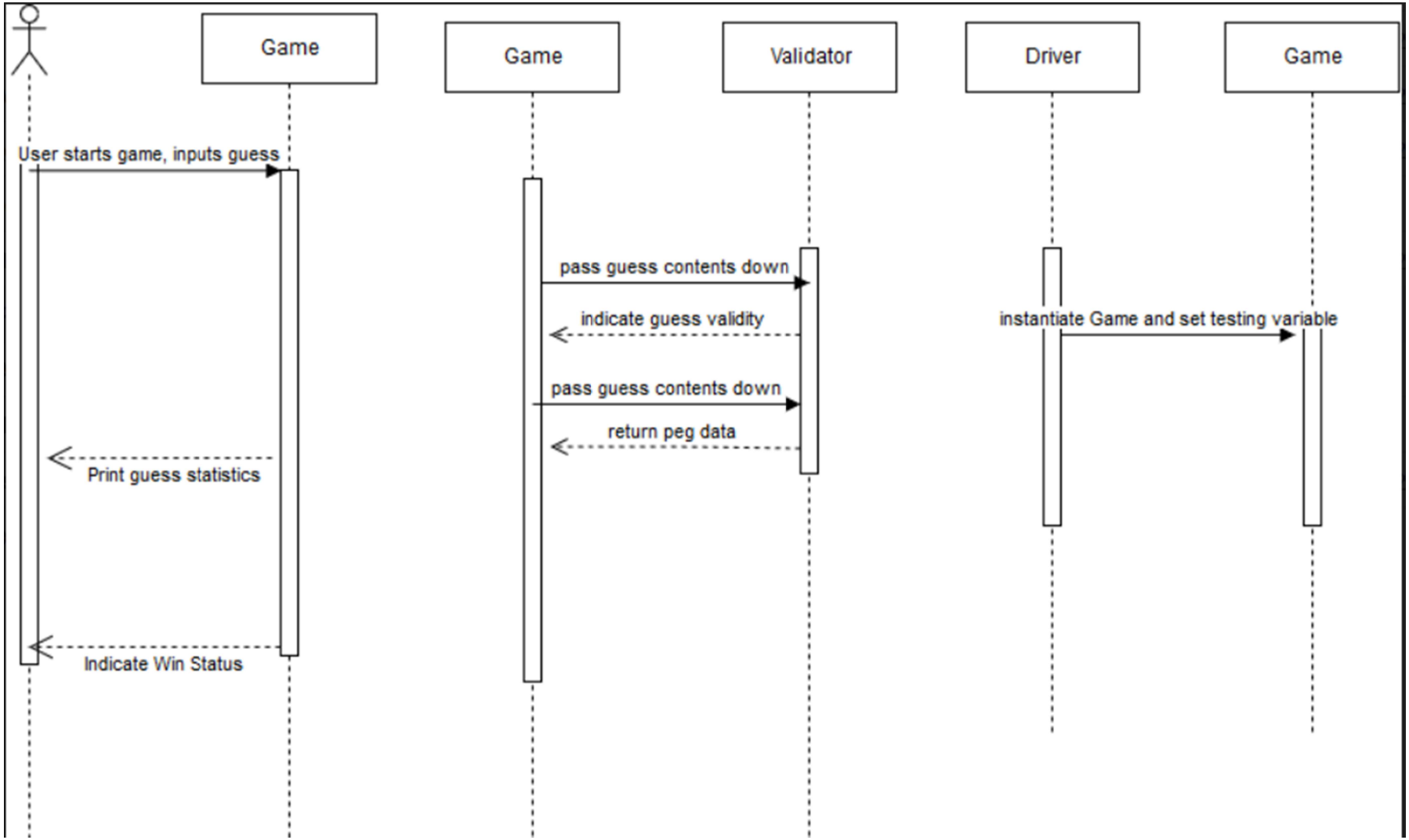
Use case: This implementation of mastermind can be used recreationally by users of any age. The users must be able to use a keyboard to start the game and input their guesses and they must also be able to read the screen to engage with the game. This game will likely appeal to younger demographics who are in a stage of developing analytical and numerical reasoning, as the goal is to present the optimal guess given abstract information about the solution. Mastermind helps develop these skills.

Class Diagram: The class diagram below implements 5 classes total. We are given GameConfiguration and SecretCodeGenerator for testing purposes. GameConfiguration allows the game to be flexible due to easily adjustable attributes and rules. SecretCodeGenerator creates solutions per each game, and defines the pseudorandom nature of the code generation.

I implemented the Driver, Game, and Validator classes. Driver houses the game as a whole and drives the program at runtime. Within driver, an instance of Game is created. The Game object manages the control flow of the entire game, including sequentially accepted games by the user. Within game, a number of external methods are needed to process, validate, and check the user input. For this, a Validator object is used. The Validator object houses a method called checkValidity that will perform the initial check regarding whether the guess is syntactically acceptable that are used per each user input. Given that the guess is acceptable, Validator uses pegCount to compare the guess and solution and process the abstract data to return to the user. In this case, pegCount returns a single String object containing the guess and peg counts to Game in an easy to output format.

Sequence Diagram: The sequence diagram below demonstrates how the User engages with the program, and where this fits with internal processes. From the user's perspective, once the game is started, they input guesses and receive feedback until the win condition is met. After this, they may choose to start a new game. While the user is waiting for feedback, the GameRunner, or the Game object, will contact the Validator object for feedback on the syntactic validity of the guess. If it is valid, the Runner will make a new request to the Validator for the peg count analysis.


Driver
+main(String[] args)
creates game object
Game
- Scanner kb
- boolean isTesting
+ Game(boolean isTesting)
- runGame()
Provides external variables defining gameplay
GameConfiguration
+ int guessNumber
+ int pegNumber
+ String[] colors
Provides solutions per each game
SecretCodeGenerator
- SecretCodeGenerator instance
- Random randomGenerator
+ getInstance()
- SecretCodeGenerator()
+ getNewSecretCode()
Checks guess valididty, returns string with Peg Counts
Validator
+ checkValidity(String guess String code, String[] colors))
+ pegCount(String guess, String code)


Game
Game
Validator
Driver
Game
User starts game, inputs guess
pass guess contents down
indicate guess validity
instantiate Game and set testing variable
pass guess contents down
return peg data
Print guess statistics
Indicate Win Status