Software Construction HS 2021

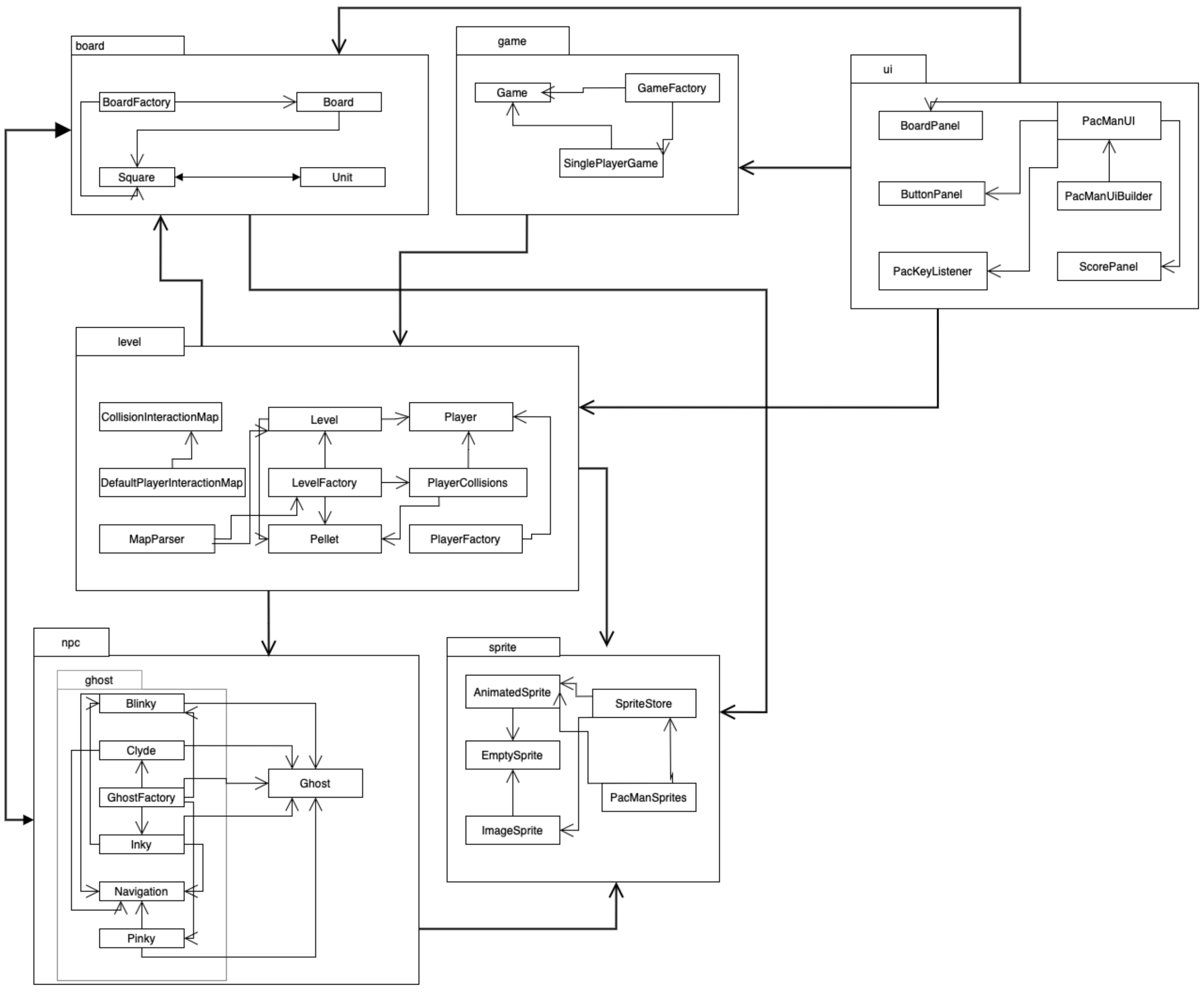
Assignment 1

Exercise 1

1. To draw the structure of the project’s packages and classes, we looked at the folders and files of the jpacman folder. This folder can be found here: main -> java -> nl -> tudelft. We decided to start there because in this folder the main packages and classes of the project can be found.

The ui-package manages all the tools a user needs to play the game. It is connected with the board-package which defines the play board. The board uses the sprite-package in order to display the players correctly.

The ui-package is also connected with the game-package which controls the start and ending of the game. This package then is connected with the level-package which handles the rules and conditions of the game. The level-package is further connected with the npc- and sprite-package. The npc-package handles the different ghost's behaviour.



board
BoardFactory
Board
Square
Unit
game
Game
GameFactory
SinglePlayerGame
ui
BoardPanel
PacManUI
ButtonPanel
PacManUiBuilder
PacKeyListener
ScorePanel
level
CollisionInteractionMap
Level
Player
DefaultPlayerInteractionMap
LevelFactory
PlayerCollisions
MapParser
Pellet
PlayerFactory
npc
ghost
Blinky
Clyde
GhostFactory
Ghost
Inky
Navigation
Pinky
sprite
AnimatedSprite
SpriteStore
EmptySprite
PacManSprites
ImageSprite

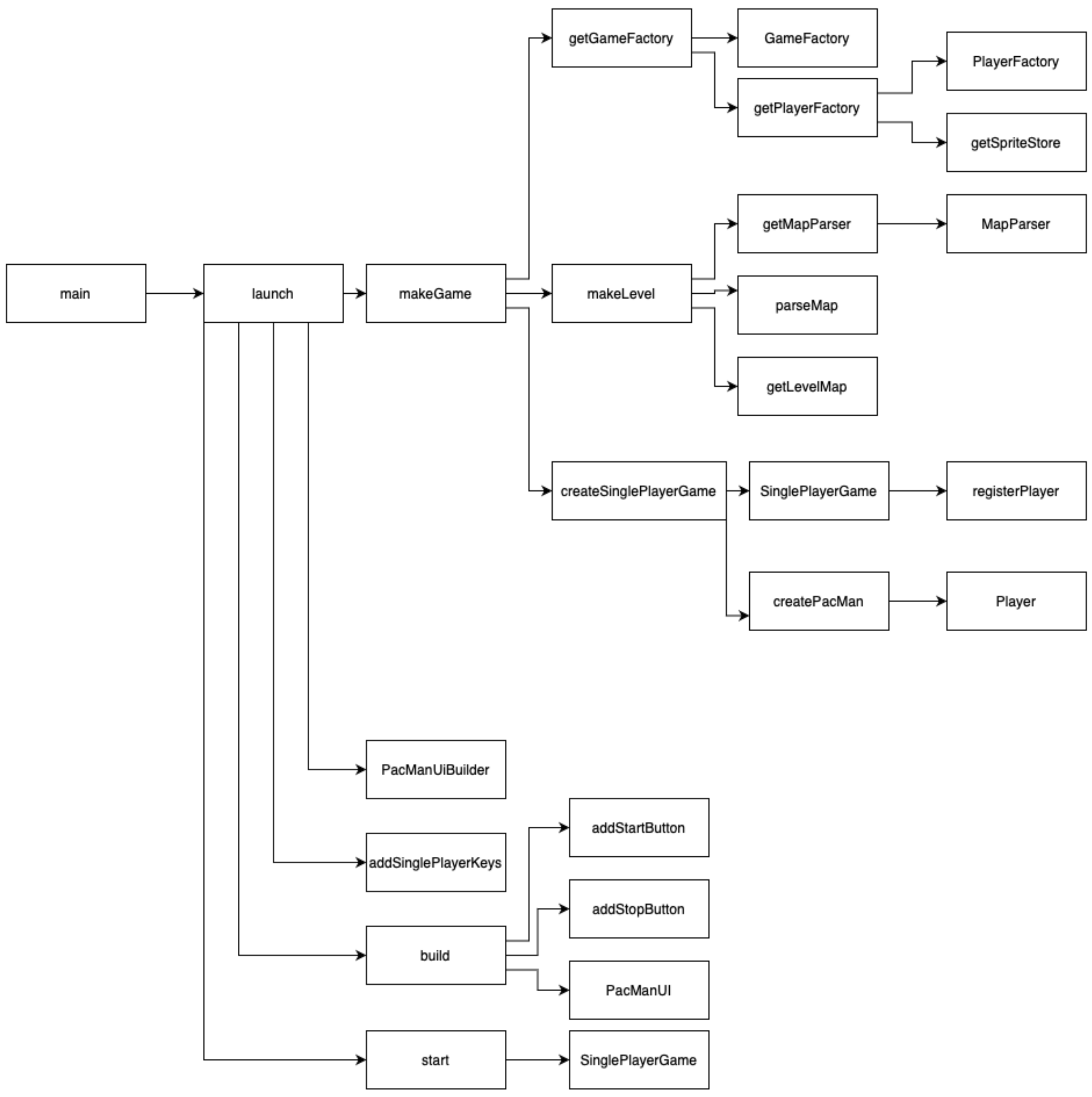
2. The call graphs entry point is the main function in the Launcher.java file. From there we explore up to 6 levels. We chose to use 6 levels because it shows the most important relationships while still keeping the call graph transparent.

The main method calls the launch Method in the Launcher class. The launch method then calls several methods in the following order:

1. makeGame, 2. PacManBuilder, 3. addSinglePlayerKeys, 4. build, 5. start

The makeGame method then calls the three methods getGameFactory, makeLevel and createSinglePlayerGame in order to create a new game. The 3 methods themself call again several methods which are needed to create the new game.

The build method then calls the methods addStartButton, addStopButton and PacManUI in order to build the UI for the above created new game. Then finally the start method of the PacManUI class is called and the game is displayed to the user.


main
launch
makeGame
getGameFactory
GameFactory
getPlayerFactory
PlayerFactory
getSpriteStore
makeLevel
getMapParser
MapParser
parseMap
getLevelMap
createSinglePlayerGame
SinglePlayerGame
registerPlayer
createPacMan
Player
PacManUiBuilder
addSinglePlayerKeys
build
addStartButton
addStopButton
PacManUI
start
SinglePlayerGame

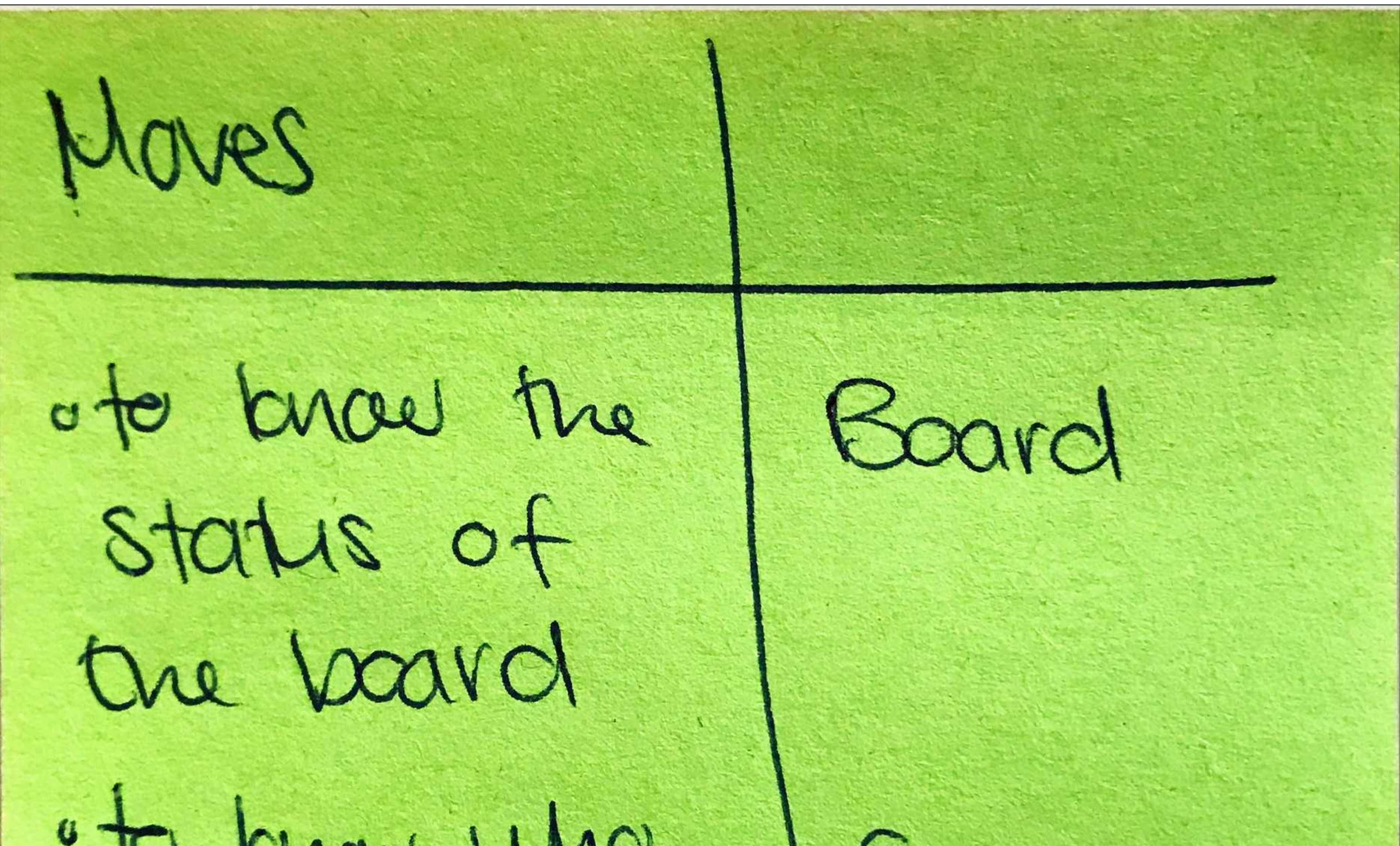
Exercise 2

1. First we read the Checker's Game Requirements and the Checkers' rules and highlighted the nouns in order to get a first idea what the classes could be. We then made the following list with the candidate classes:

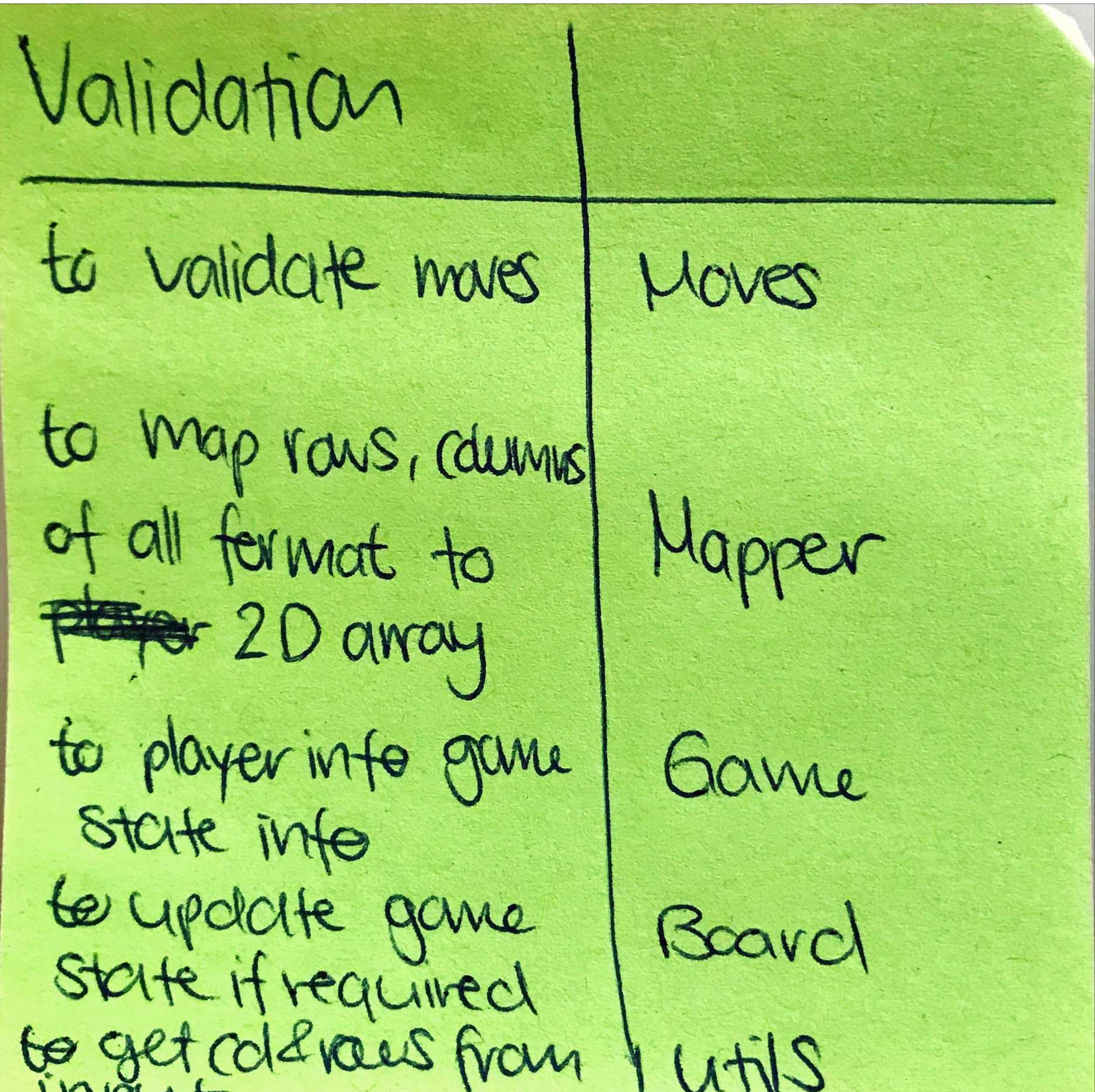
   List of Candidate Classes

   - Board
   - Users
   - Moves
   - Input
   - Pawn
   - King
   - Validity of the move
   - Winner
   - Simple move
   - Single jump move
   - Multiple jump move
   - Round

   For each of these candidate classes we designed a CRC Card and defined its responsibilities. In order to do this we looked again at the text and the verbs corresponding to the previously highlighted nouns. We then discussed several scenarios in order to find relations between the classes and noted them on the CRC cards in the collaboration column. During the discussion we kicked some of the classes out or merged them with other classes.



Moves	
• to know the status of the board	Board
• to know who is playing (red or white)	Game

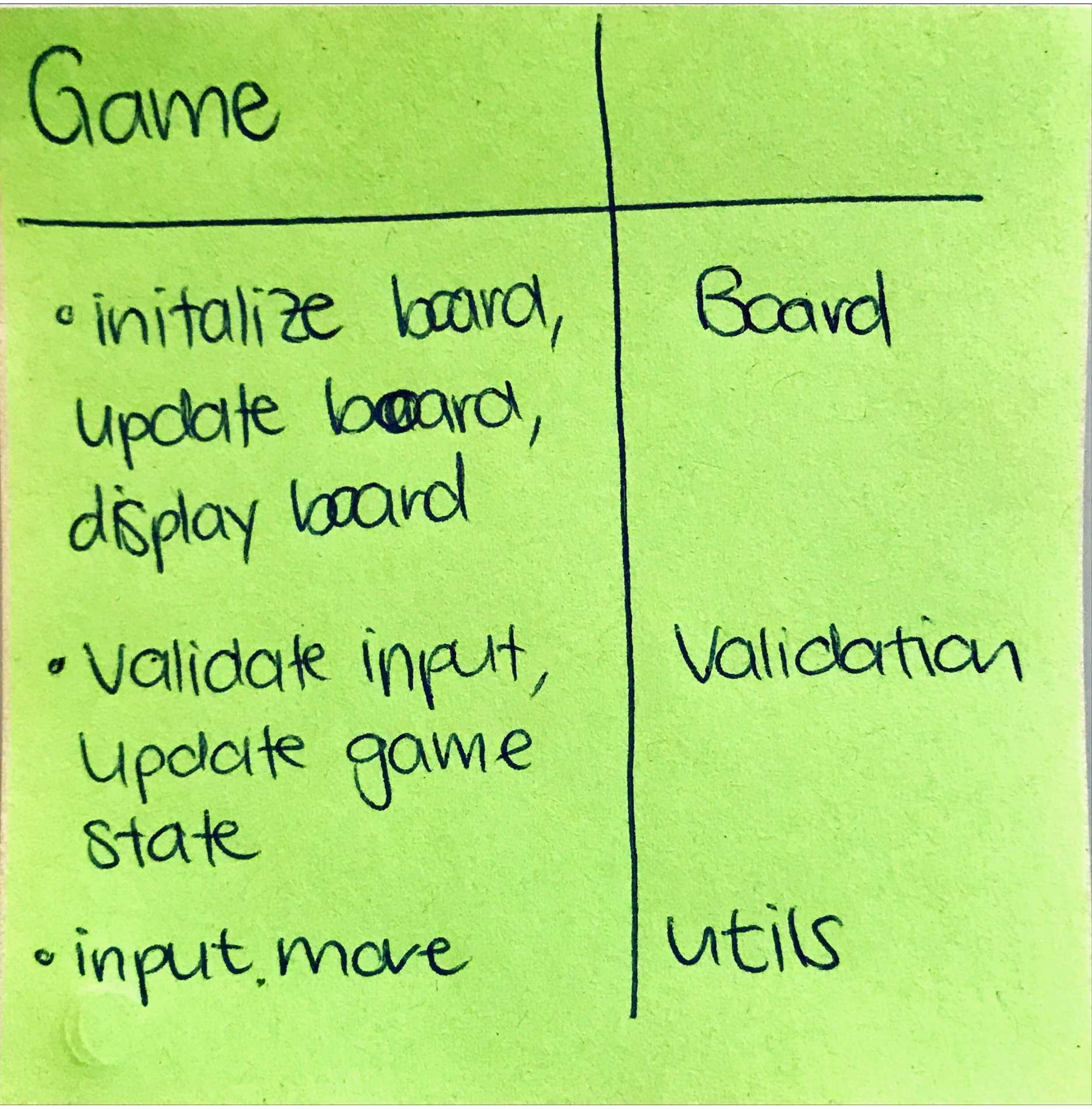


Validation	
to validate moves	Moves
to map rows, columns of all format to ~~player~~ 2D array	Mapper
to player info game state info	Game
to update game state if required	Board
to get col&rows from input	Utils

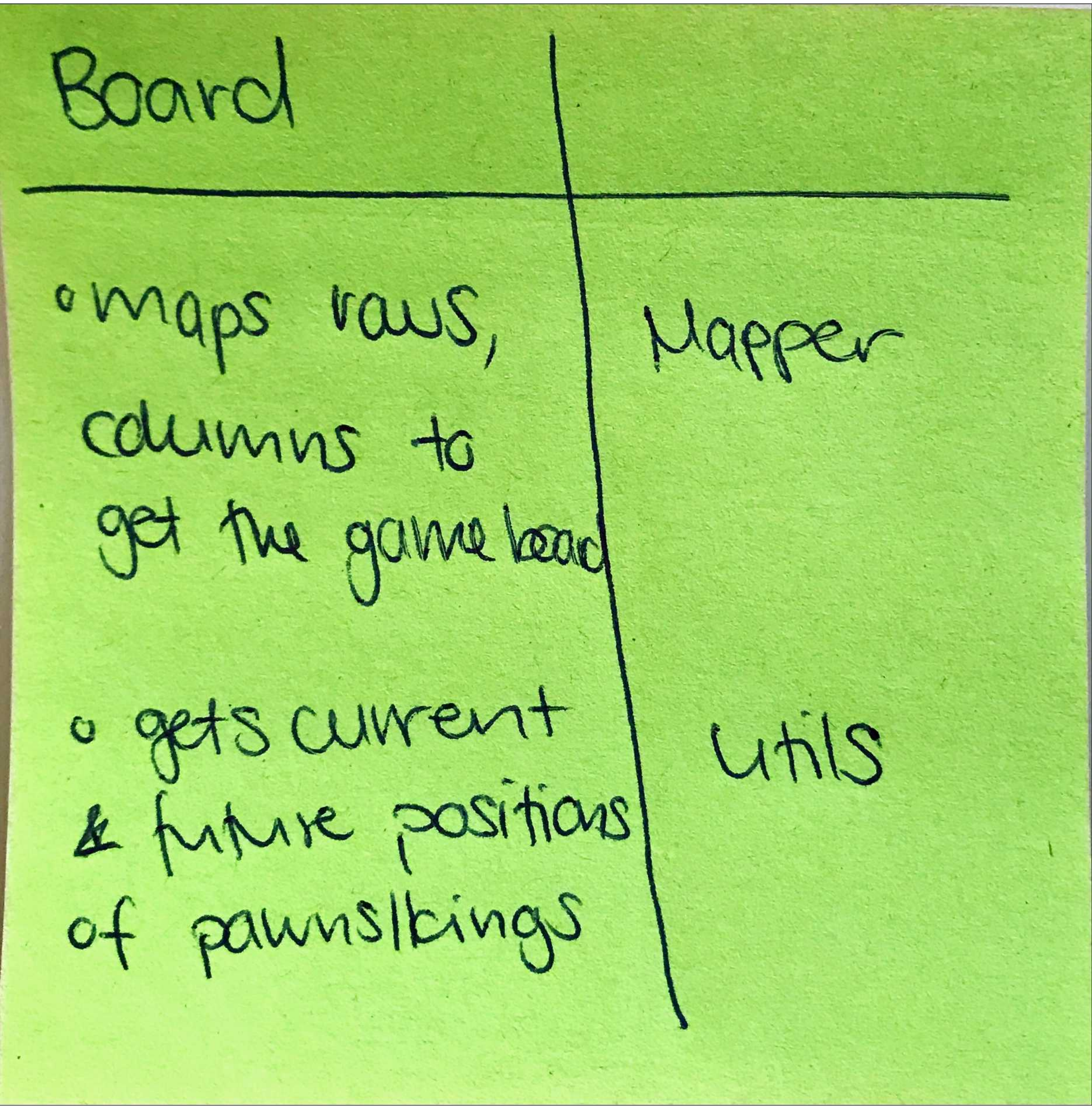
Utils

- get current and future positions
- get input from player

Mapper	
• to map rows & columns from board to 2D array to make callculations / use values later	



Game	
• initalize board, update board, display board	Board
• Validate input, update game state	Validation
• input move	utils



Board	
◦ maps rows, columns to get the game board	Mapper
◦ gets current & future positions of pawns/kings	Utils

2. Our main classes are Game, Board, Validation and Moves.

   The Game-class handles the input of the players and provides the UI. The UI consists of the game board, the input field and some messages. In order to initialize and update the game board, the Game-class calls the Board-class. In order to update the game board, the Board-class calls the helper classes utils and Mapper.

   The Game-class is further used to validate the input and to update the game state. In order to validate the move entered by the player, it calls the Validation-class. The Validation-class then checks the format of the move and calls the Moves-class in order to check if the entered move is in the list of possible moves we get from the Moves-class.

   The Move-class gets for a given move all possible moves. It handles all the rules of the moves. To do so it needs some information from the Game-class and from the Board-class.

3. The User- and Input-class technically do the same, since there is no other information about the users than the move they entered. So we decided to put them together in the helper class called utils. This class is then called by the other classes in order to handle the input.

   The Round-class is quite similar to the Board-class. It simply updates the Board defined in the Board-class. So we decided to put the functionality of the Round-class into the Board-class.

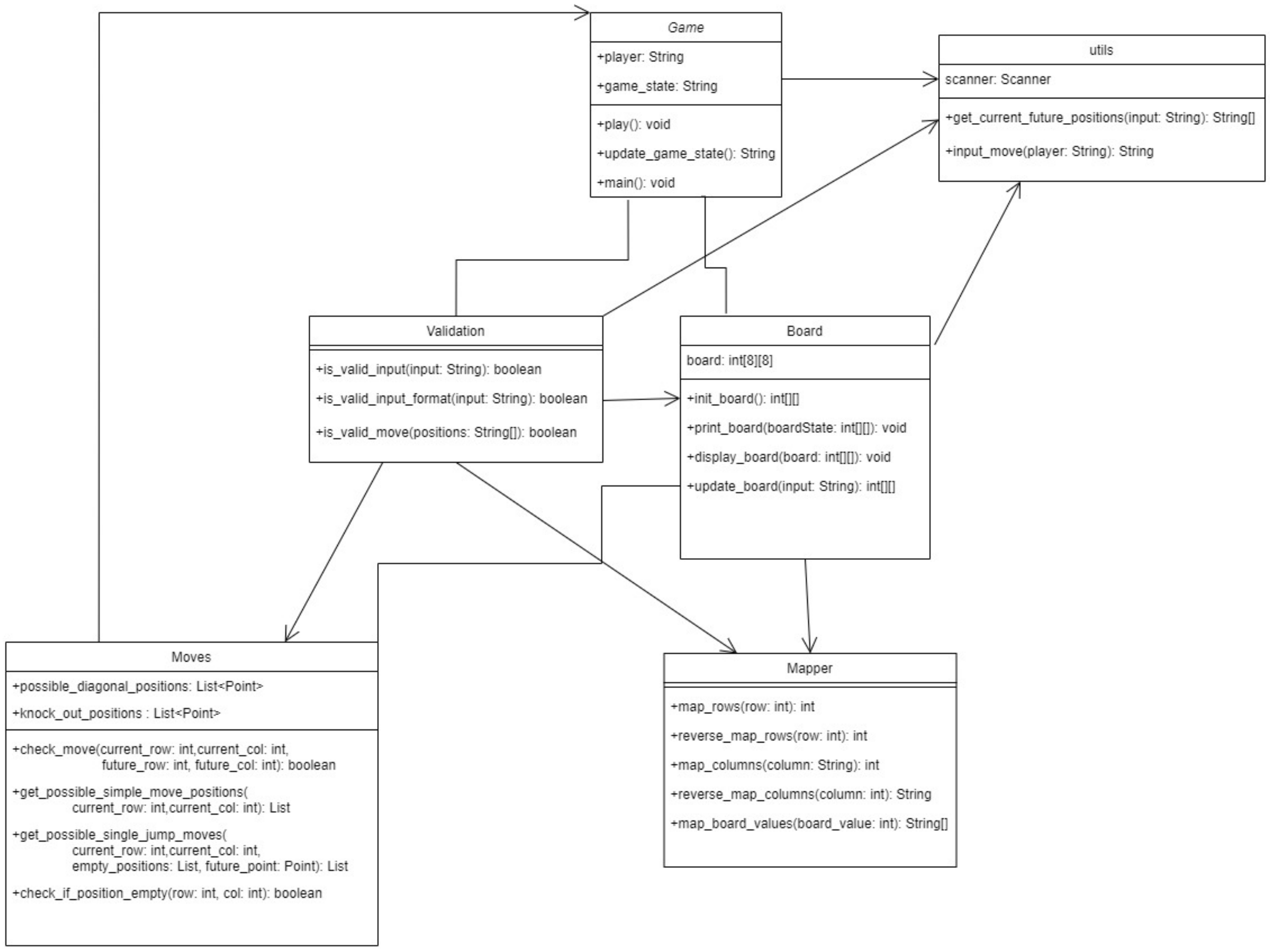
   The King- and Pawn-class are of no particular use since we store the positions of the pieces in the Board-class. To update a pawn to a king we decided to use the Board-class since there we already update the positions and type of the pices.

   SimpleMove-, SingleJumpMove- and MultipleJumpMove-class are not needed since the rules for the moves are all handled in the Moves-class.

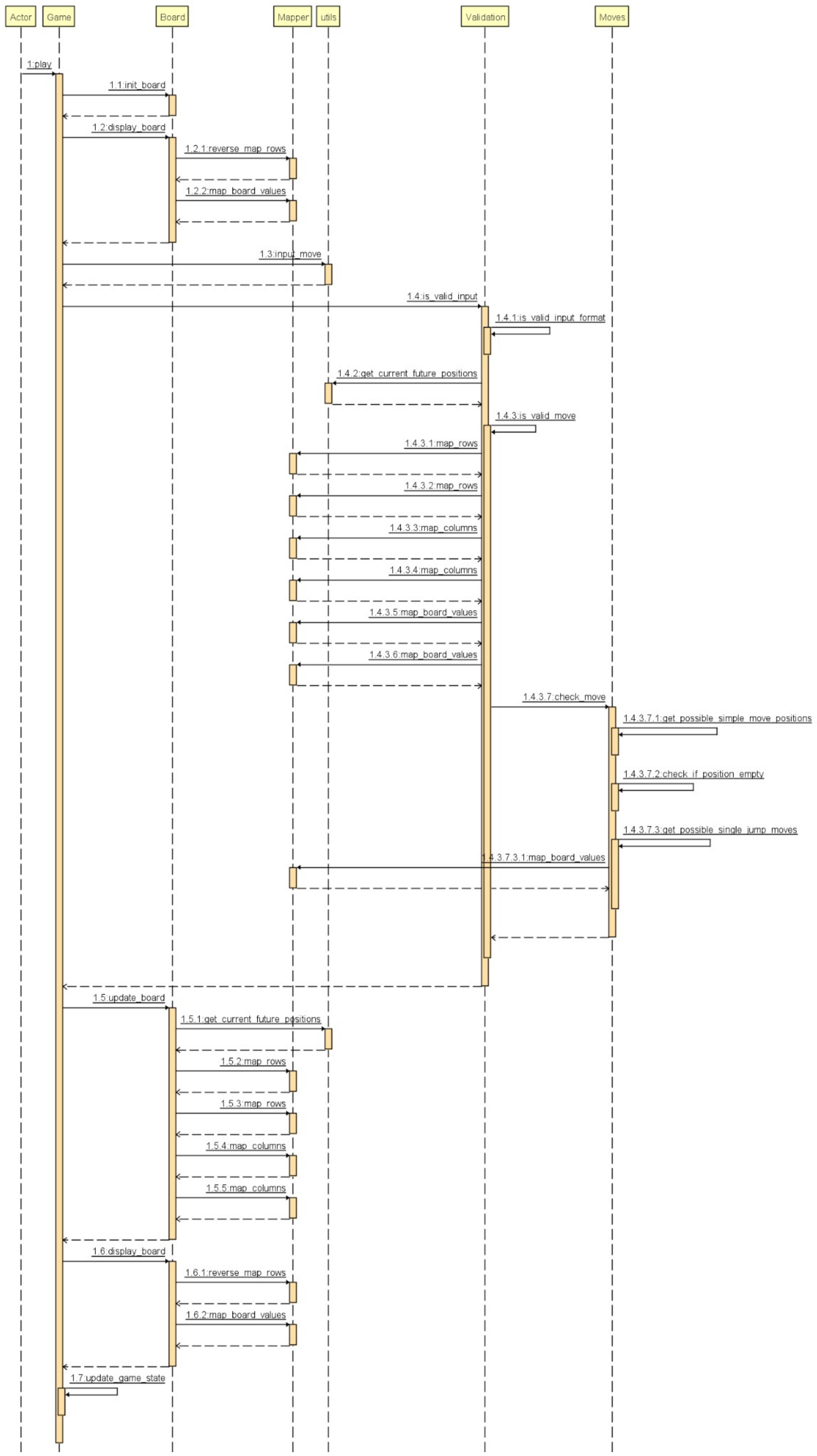
   The Winner-class, where we initially wanted to check if the game is over or if it continues, was dropped and we have put this task into the Game-class.

   We added two helper classes. The utils-class reads the input from the user and gets the current and future positions of the pieces. The Mapper-class helps to map the rows and columns from the board to a 2D-array which is used later to work with it. These two classes could as well be one class since they both provide smaller helping methods to the rest of the code.

4. Class Diagrams


Game
+player: String
+game_state: String
+play(): void
+update_game_state(): String
+main(): void
utils
scanner: Scanner
+get_current_future_positions(input: String): String[]
+input_move(player: String): String
Validation
+is_valid_input(input: String): boolean
+is_valid_input_format(input: String): boolean
+is_valid_move(positions: String[]): boolean
Board
board: int[8][8]
+init_board(): int[][]
+print_board(boardState: int[][]): void
+display_board(board: int[][]): void
+update_board(input: String): int[][]
Moves
+possible_diagonal_positions: List<Point>
+knock_out_positions : List<Point>
+check_move(current_row: int,current_col: int, future_row: int, future_col: int): boolean
+get_possible_simple_move_positions(current_row: int,current_col: int): List
+get_possible_single_jump_moves(current_row: int,current_col: int, empty_positions: List, future_point: Point): List
+check_if_position_empty(row: int, col: int): boolean
Mapper
+map_rows(row: int): int
+reverse_map_rows(row: int): int
+map_columns(column: String): int
+reverse_map_columns(column: int): String
+map_board_values(board_value: int): String[]

5. Sequence Diagrams


Actor
Game
Board
Mapper
utils
Validation
Moves
1:play
1.1:init_board
1.2:display_board
1.2.1:reverse_map_rows
1.2.2:map_board_values
1.3:input_move
1.4:is_valid_input
1.4.1:is_valid_input_format
1.4.2:get_current_future_positions
1.4.3:is_valid_move
1.4.3.1:map_rows
1.4.3.2:map_rows
1.4.3.3:map_columns
1.4.3.4:map_columns
1.4.3.5:map_board_values
1.4.3.6:map_board_values
1.4.3.7:check_move
1.4.3.7.1:get_possible_simple_move_positions
1.4.3.7.2:check_if_position_empty
1.4.3.7.3:get_possible_single_jump_moves
1.4.3.7.3.1:map_board_values
1.5:update_board
1.5.1:get_current_future_positions
1.5.2:map_rows
1.5.3:map_rows
1.5.4:map_columns
1.5.5:map_columns
1.6:display_board
1.6.1:reverse_map_rows
1.6.2:map_board_values
1.7:update_game_state