Justification for Handling State

Players : I chose to store this data in the controller Class, since players have access to workers, we will be accessing their objects very frequently. Controller, being a class which hosts move player and addBlock functionalities, seems very suitable.

Current Player : This variable is stored locally within a function in the controller and printed/displayed as and when a change of player is observed. Once the role changes, it switches to the next player. Hardcore state for this player is not required.

Worker Location : The location of the worker would be feasible to add in the worker object themselves. And calling them as and when required. This would avoid confusions in code analysis.

Towers : I am maintaining the tower in terms of blocks and block height → and assigning them to the Tile Class object. A tile is a spot on the grid (A grid has 5x5 = 25 tiles). If blockHeight == 4, then the tower is inaccessible.

Winner : The winner of the game is returned and stored in the game class as well as controller class (to exclude him in the next round)

I chose this pattern because it involves using just variables rather than objects for storing data values. This would save on the memory aspect of it and also avoids multiple creation of objects. I considered creating 2 nested classes within the class Player, for the 2 workers. But that would complicate the code base and involve a lot of coupling. Instead I created one worker class and created objects of them in my player class.

Another initial thought was to use a Block/Tower class under the interface Entity, which would have players and workers in it. But that approach again would mean creating a block object for each of the spots it visits, assigning the height or overwriting that with another block object. This would lead to redundancy and wastage of memory. Hence figured we could just use a variable which contains the height of the block and store it in the tile itself.