Card object

* Category
  + **“Construction”** or “**creature”**
* Name
  + String name
* Cost (type ResourceValue)
  + E.g. {“resin”: 10, “twig”: 0, “pebble”: 3}
  + Object with fields resin, twig, pebble all ints
  + These ResourceValue objects can then be more easily added/subtracted to/from the users resources, by implementing a ResourceValue.subract(ResourceValue) and .add() methods between two cost objects, with error codes returned if the cost would “overdraw” the users resources
* Use\_effect() a function that is triggered on use or upon the players command, and executes the effect of that card, e.g. getting resources, stealing another players cards, etc.
* Effectively in the state vector, each card has an owner, and an effect that can be triggered, some take arguments, in which case you have multiple possible actions which represents the permutations of ways in which you could execute that effect function too. (any illegal action will be rejected by the game, forcing the AI to choose again, i.e. 0 cost/return environment iteration and sends the AI back to the action selection)
  + From the AIs perspective, each card will have an owner of “ownername” or None if it has not been revealed yet. Also each player will have a num\_cards\_in\_hand field – and the total number of cards in players’ hands will be greater than the number of revealed cards, since players hands are secret.

Workers

Class Worker

* Owner (string) – name of the player that owns the worker
* Each player gets a number of workers each round and placing them on a
* When playing do:

If (ActionSpaces[x].getWorker() == None && Player.getWorker()) {

ActionSpaces[x].addWorker(Player.getWorker())

Player.removeWorker()

}

Player.getWorker() gets the first worker in the player workers list

Player.removeWorker() pops the first worker from the player workers list

ActionSpace.addWorker(Worker) adds that worker to the action space

* Note: In the game state vector, we will have to list all of the action spaces, including: what resources they have, and the worker object that is on there (or None is there isn’t a worker on it yet)
  + Also remember some action spaces can have 2 slots in them, so it will need to be a list of possible workers on there
  + Also note if there are double slots, you cannot place your own workers on both of the slots (a player can only use up to one slot on a dual slot ActionSpace)

Action Spaces

class ActionSpace

* Value (type ResourceValue)
* Workers (type Worker[]) (list of workers or Nones to show what is taking up each of the slots on the ActionSpace)

Basic action spaces where you can move your workers