General stuff:

Use model.waitingForAction to receive a boolean to know if the model is currently in a +action> state.

For each player, map the opponents’ name to player 1, player2 and player3 for usage in production rules.

The first act will happen automatically when receive “first\_start” state. Then the state of the model will be changed to “gofish” or “succeed”. The state of other players will be changed to “wait\_memorize”

This is a rough outline for the action that will have to be taken depending on the model’s state

### First\_start:

Give the model the state = first\_start and others the wait\_memorize **Discuss?**

Checked with the waitingForAction == True and lastAction(“state”) == first\_start

Step 1: Obtain info from model about decision

Var = model.lastAction(“card\_ask”) //Card being asked

Var2 = model.lastAction(“opponent\_player”) //Opponent being asked

Step 2: perform action and get result

Step 3: return result to model

model.modifyLastAction(“card\_ask”, var)

model.modifyLastAction(“card\_get”, result) //On success =var, on fail = gofish

model.modifyLastAction(“current\_player”, self)

model.modifyLastAction(“opponent\_player”, var2)

model.modifyLastAction(“round\_state”, result) //succeed/gofish

model.modifyLastAction(“state”, result) //same as above

### Not your round:

This happens whenever its not the models round so they’re just storing stuff

Checked with the waitingForAction == True and lastAction(“state”) == wait\_memorize

Whenever someone asks for a card, give result to models

model.modifyLastAction(“card\_ask”, var) //Same stuff

model.modifyLastAction(“card\_get”, result) //=var or gofish

model.modifyLastAction(“current\_player”, player) //whoever has the turn right now

model.modifyLastAction(“opponent\_player”, var2) //whoever is being asked, possible transform

this into “self” whenever it is asking that model? **Discuss**

model.modifyLastAction(“round\_state”, result) //succeed/gofish

model.modifyLastAction(“state”, result) //same as above

### Ask:

Checked with the waitingForAction == True and lastAction(“state”) == ask

Var = model.lastAction(“card\_ask”) //Card being asked

Var2 = model.lastAction(“opponent\_player”) //Opponent being asked

model.modifyLastAction(“card\_ask”, var)

model.modifyLastAction(“card\_get”, result) //On success =var, on fail = gofish

model.modifyLastAction(“current\_player”, self)

model.modifyLastAction(“opponent\_player”, var2)

model.modifyLastAction(“state”, result) //succeed/gofish

### Checking:

Checked with the waitingForAction == True and lastAction(“state”) == checking

strat = model.lastAction(“strategy”) //Strategy used, redundant

Switch case for model.lastAction(“card\_deck”)

Case 1: (card\_deck = first)

Retrieve the first card in the hand of the active model

Case 2: (card\_deck = =card)

Retrieve the card after the specified card (index +1, if index is out of bounds, return checking\_failed)

Case 3: (card\_deck = multiple)

Retrieve the first card type from the active model’s hand that has multiples in the hand. If there are no multiples, return the first card. (can never go checking\_failed)

model.modifyLastAction(“card\_deck”, result of above) //Either a card or true/nil ?

model.modifyLastAction(“strategy”, strat) //Redundant

model.modifyLastAction(“state”, result switch case) //checking on successful check, checking\_failed on a failed check which is when the requested card was not in or multiples were not found.