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

Multiple models: Short assumption for how this will work is that we basically just have a model1, model2 and model3 in the form of:

var model1: go\_fish1?

var model2: go\_fish2?

var model3: go\_fish3? (Example)

These would be 3 different act-r files with basically the same code in them

Switch 2,3,4 players

Case 2: load go\_fish2p

Case3: load go\_fish1 3p, gofish2 3p

Case 4:load go\_fish1 4p, go\_fish2 4p, go\_fish3 4p (Example)

### First\_start:

Give the model the state = first\_start

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

Step 1: Obtain info from model about decision

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

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

Step 2: perform action and get result

Step 3: return result to model

model!.modifyLastAction(slot: “card\_ask”, value: String(var))

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

model!.modifyLastAction(slot: “current\_player”, value: String(self))

model!.modifyLastAction(slot: “opponent\_player”, value: String(var2))

model!.modifyLastAction(slot: “first\_starter”, value: String(true/false)) //Mention if this is true or false

model!.modifyLastAction(slot: “state\_round”,value: String(result)) //succeed/gofish

model!.modifyLastAction(slot: “state”, value: String(result)) //wait\_memorize/succeed/gofish/start

### Not your round:

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

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

Whenever someone asks for a card, give result to models

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

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

model.modifyLastAction(slot: “card\_ask”, value: String(var)) //Same stuff

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

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

model.modifyLastAction(slot: “opponent\_player”, value: String(var2)) //make sure to map correctly

model.modifyLastAction(slot: “state\_round”, value: String(result)) //succeed/gofish

model.modifyLastAction(slot: “state”, value: String(result)) //same as above

### Ask:

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

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

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

model.modifyLastAction(slot: “card\_ask”, value: String(var))

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

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

model.modifyLastAction(slot: “opponent\_player”, value: String(var2))

model.modifyLastAction(slot: “set”, value: String(true/false)) //return true/false

model.modifyLastAction(slot: “state”, value: String(result)) //succeed/gofish

### Checking:

Checked with the model1!.waitingForAction == True and lastAction(slot: “state”) == checking

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

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

Case 1: (slot: “card\_deck” == first)

Retrieve the first card in the hand of the active model

Case 2: (slot: “card\_deck” == =card)

Boolean to determine if the =card is in the active model’s hand

Case 3: (slot: “card\_deck” == multiple)

Retrieve the first card type from the active model’s hand that has multiples in the hand

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

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

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