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.

Also, when receiving the “ask” state, inverse map the received player info back to the swift player name!

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” or “start” depends on the round state and player order.

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

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

Step 1: Obtain info from model about decision

Only have the state “first\_start” state.

Step 2: perform action and get result:

1. Choose a player to be the first starter and change parameter “first\_starter” = true.
2. For the first starter, ask his “player1” for his first card.
3. Based on the result, change every models’ states (“wait\_memorize” or “start” or “gofish”/”succeed”)

Step 3: return result to model

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

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

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

model.modifyLastAction(“state”, result) //(“wait\_memorize” or “start” or “gofish”/”succeed”)

model.modifyLastAction(“first\_starter”, result)// only change this state of one model!!!

### 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

The information is from the round

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

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

model.modifyLastAction(“state”, result) // Based on the result, change every models’ states (“wait\_memorize” or “start”)

### 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(“state”, result) //succeed/gofish

### Checking:

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

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 = a certain card)

Retrieve the card after the specified card (index +1 (for unique list), 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 nil model.modifyLastAction(“state”, result switch case) //checking on successful check, checking\_failed on a failed check which is when the requested card has reached the end

***Original:***

Swift/Act-r communication through action module:

* ***Situation: The first round for the player.***
  + Input:
    - State: **first\_start**
  + Output:
    - Card\_ask
    - Current\_player
    - Opponent\_player
    - first\_starter (default false)
    - **State: state of self (wait\_memorize /gofish/succeed/start)**
  + Action outside production rules:
    - One random player will be selected to take the first act (ask the player to his left for the first card at hand). Change the first\_starter to true for the first player.
    - The first act will happen automatically when receive “first\_start” state. Then the state of the player will be changed to “gofish” or “succeed”. The state of other players will be changed to “wait\_memorize”. If the state of this round is gofish, one player will be changes to “start”.
    - When the round state is “**gofish**”, swift logic needs to decide who is the next to player and change players’ states accordingly
* ***Situation: Not your round.*** 
  + Input:
    - State: **wait\_memorize**
  + Output:
    - Card\_ask
    - Current\_player
    - **State: wait\_memorize (next round is still wait), start (you will initiate the next round --- go to the strategy production rules)**
  + Action outside production rules:
    - When the round state is “**gofish**”, swift logic needs to decide who is the next to player and change players’ states accordingly.
  + Actions in act-r:
    - Creating memory chunks for player/card relationship
* ***Situation: ask a player for a card.*** 
  + Input:
    - State: **ask**
    - Card\_ask
    - Opponent\_player
  + Output:
    - State: state of self (gofish/succeed)
  + Actions in act-r:
    - Creating memory chunks for exposed cards
* ***Situation: check card at hand.*** 
  + Input:
    - State: **checking**
    - card\_deck: first (ask for first card) or card var (ask for the card after card var) or multiple (check for multiple cards)
  + Output:
    - card\_deck (in the “multiple” case, if no multiple cards, send back the first card)
    - State: checking or checking\_failed (reached the last card)
* Other interactions:
  + For each player, map the opponents’ name to player 1, player2 and player3 for usage in production rules.