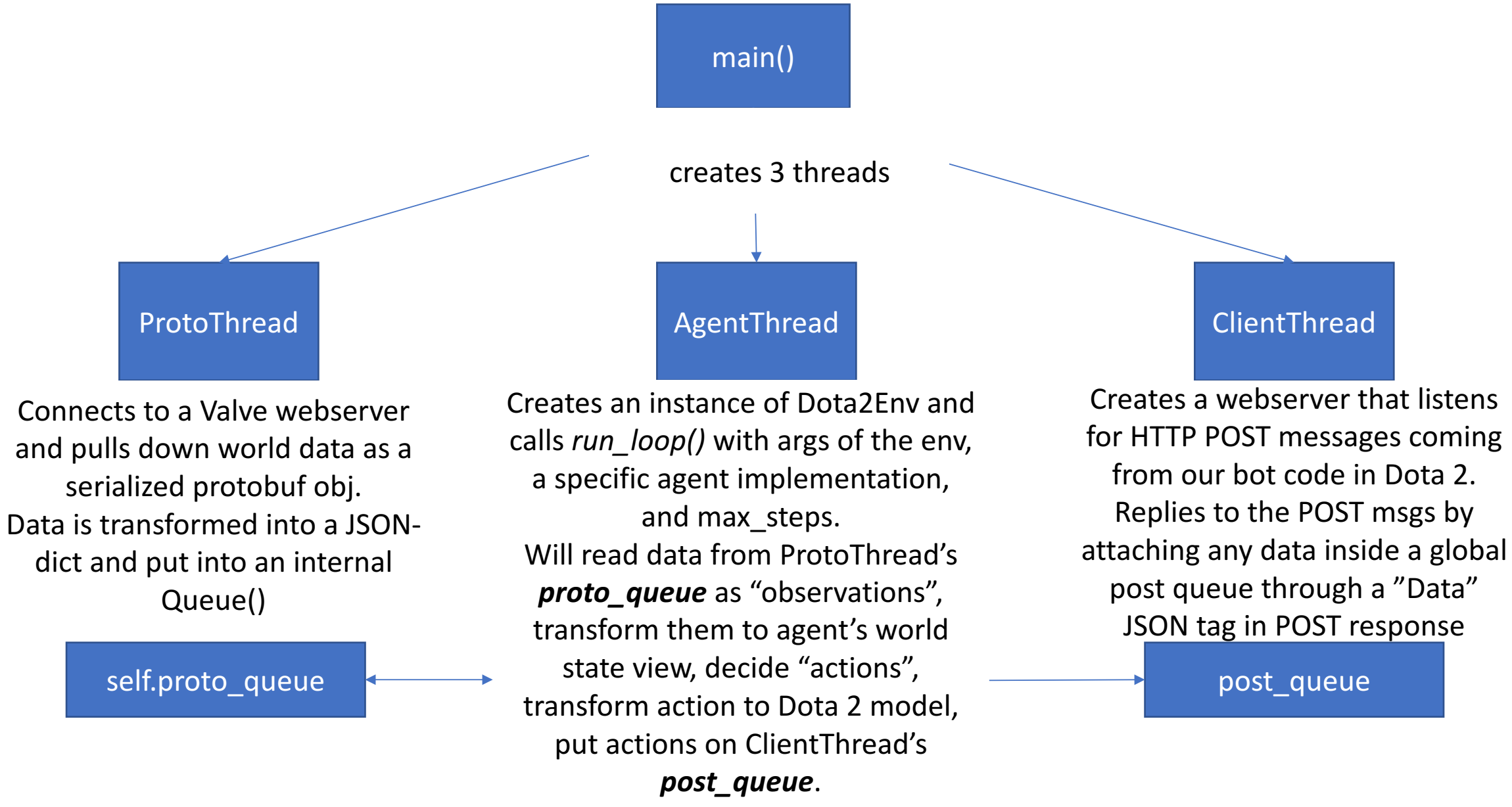
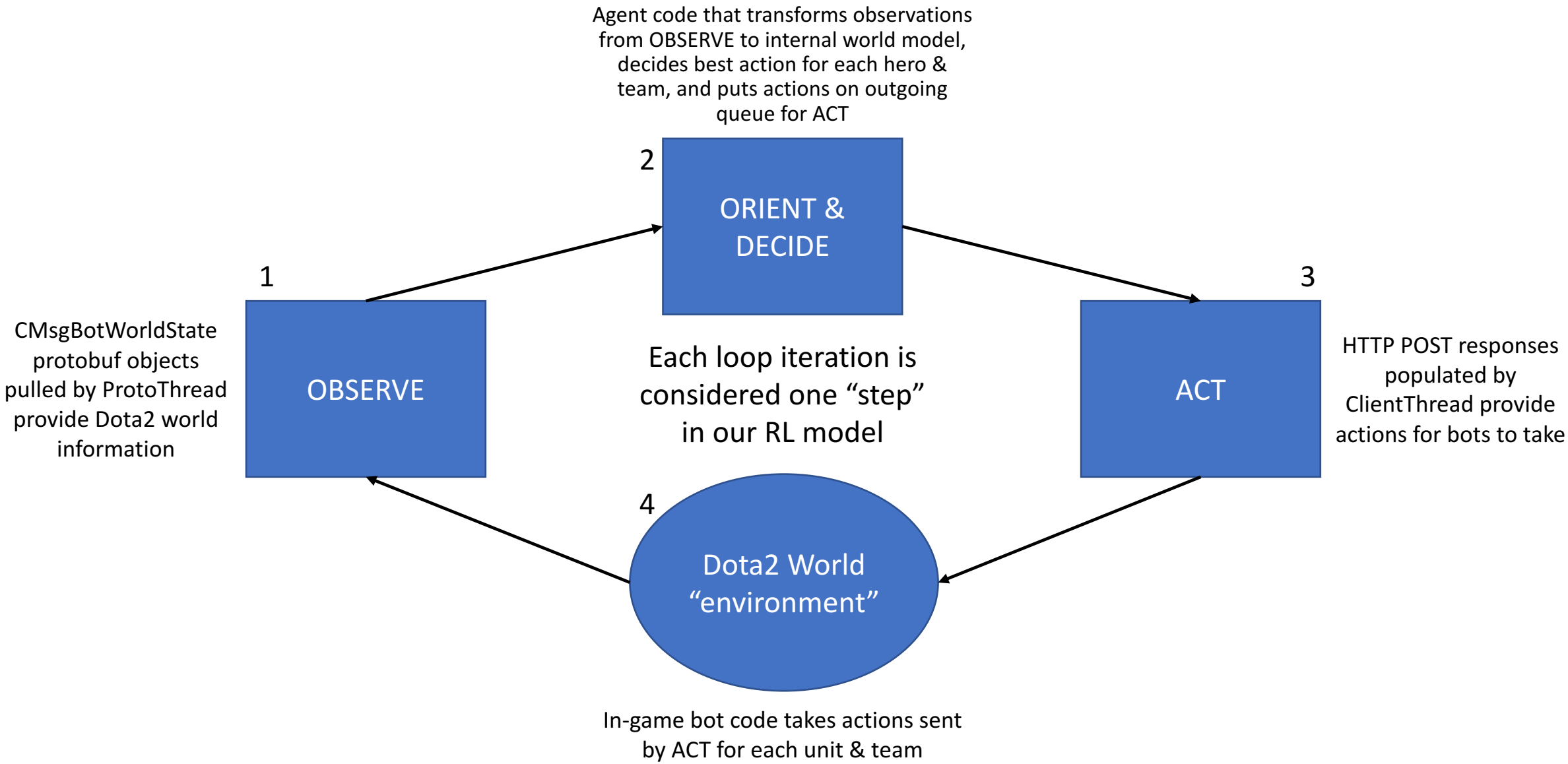
pydota2/bin/human_play.py
Data Flow


main()
creates 3 threads
ProtoThread
AgentThread
ClientThread
Connects to a Valve webserver and pulls down world data as a serialized protobuf obj.
Data is transformed into a JSON-dict and put into an internal Queue()
Creates an instance of Dota2Env and calls *run_loop()* with args of the env, a specific agent implementation, and max_steps.
Will read data from ProtoThread's ***proto_queue*** as "observations", transform them to agent's world state view, decide "actions", transform action to Dota 2 model, put actions on ClientThread's ***post_queue***.
Creates a webserver that listens for HTTP POST messages coming from our bot code in Dota 2.
Replies to the POST msgs by attaching any data inside a global post queue through a "Data" JSON tag in POST response
self.proto_queue
post_queue

run_loop.py


Agent code that transforms observations from OBSERVE to internal world model, decides best action for each hero & team, and puts actions on outgoing queue for ACT
2
ORIENT & DECIDE
1
3
CMsgBotWorldState protobuf objects pulled by ProtoThread provide Dota2 world information
OBSERVE
Each loop iteration is considered one “step” in our RL model
ACT
HTTP POST responses populated by ClientThread provide actions for bots to take
4
Dota2 World “environment”
In-game bot code takes actions sent by ACT for each unit & team

run_loop.py - Data Flow

pydota2/env/run_loop.py

```
while True:
  total_frames += 1
  actions = [agent.step(timestep)
             for agent, timestep in zip(agents, timesteps)]
  if max_frames and total_frames >= max_frames:
    return
  if timesteps[0].last():
    break
  timesteps = env.step(actions)
```

pydota2/agent/<chosen_agent>.py : *example random_agent.py*

```
def step(self, obs):
  super(RandomAgent, self).step(obs)
  function_id = numpy.random.choice(obs.observation["available_actions"])
  args = [[numpy.random.randint(0, size) for size in arg.sizes]
                  for arg in self.action_spec.functions[function_id].args]
  return actions.FunctionCall(function_id, args)
```

pydota2/agents/base_agent.py

```
def step(self, obs):
  self.steps += 1
  self.reward += obs.reward
  return actions.FunctionCall(0, [])
```

pydota2/env/available_actions_printer.py

```
def step(self, *args, **kwargs):
  all_obs = super(AvailableActionsPrinter, self).step(*args, **kwargs)
  for obs in all_obs:
    for avail in obs.observation["available_actions"]:
      if avail not in self._seen:
        self._seen.add(avail)
        self._print(self._action_spec.functions[avail].str(True))
  return all_obs
```

pydota2/env/dota2_env.py

```
def step(self, actions):
    """Apply actions, step the world forward, return observations."""
    if self._state == environment.StepType.LAST:
        return self.reset()

    # send each agent action to the dota2 client bot(s)
    self._parallel.run(
        (c.add_to_post_queue, self._features.transform_action(o.observation, a))
        for c, o, a in zip(self._post_controller, self._obs, actions)
    )

    self._state = environment.StepType.MID
    return self._step()
```

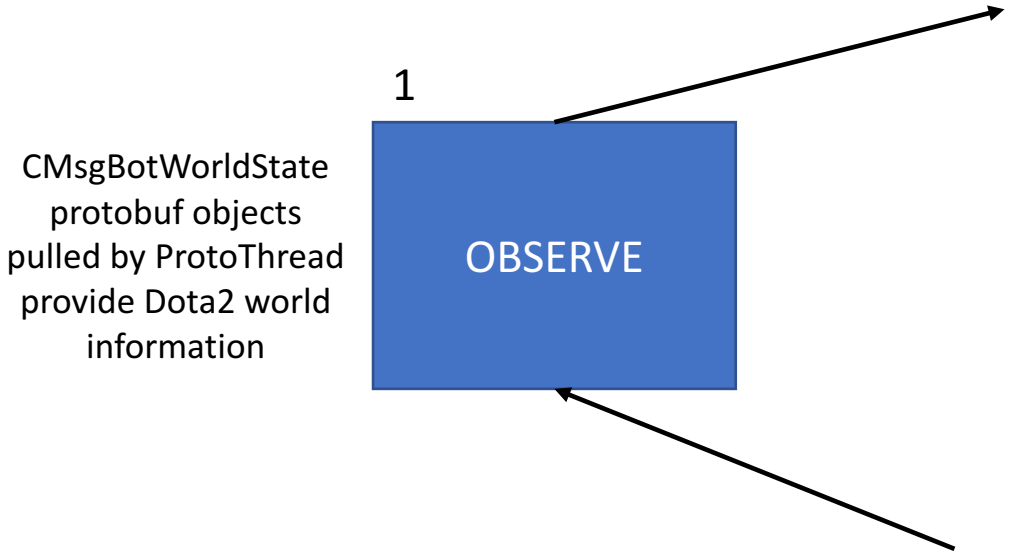
on next page:

self._step() waits for actions to be taken by bots in Dota2 world and new observations to be made through protobuf dump; updates the current reward & discount

NOTE: Missing Concept

- This whole document starts with the run_loop essentially, which assumed we are already in-game ready to play, but what happens first is “hero selection”
- Hero Selection can also be a RL learned behavior and can heavily influence your chances of winning/losing the game
- For this document we simply assume it’s “taken care of” (/end hand wave)

Component: OBSERVE



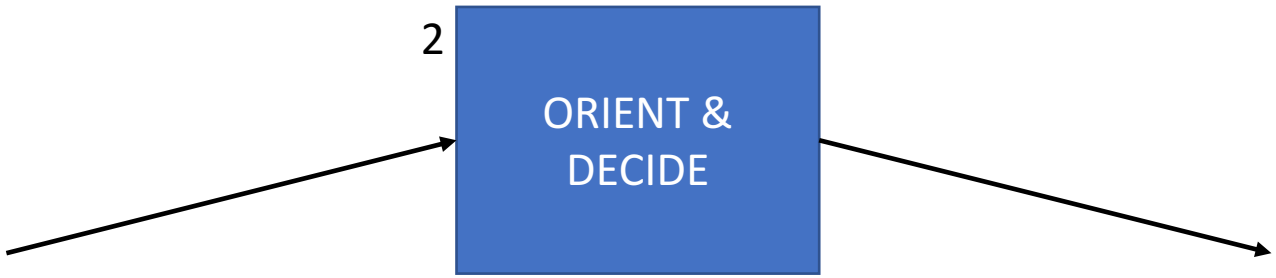
Status: COMPLETED

Issues:

- No “final” CMsgBotWorldState protobuf for game currently exists signifying a termination of game and results; also ideally game summary info
- Protobuf format could be improved by Valve
  - No need for “flying_courier” (implicit on time now)
  - Need “currentAction” and “actionArgs” for Units mapped against Valve’s ACTION TYPES
    - https://developer.valvesoftware.com/wiki/Dota_Bot_Scripting#Action_Types

Component: ORIENT & DECIDE

Agent code that transforms observations from OBSERVE to internal world model, decides best action for each hero & team, and puts actions on outgoing queue for ACT



Status: IN PROGRESS

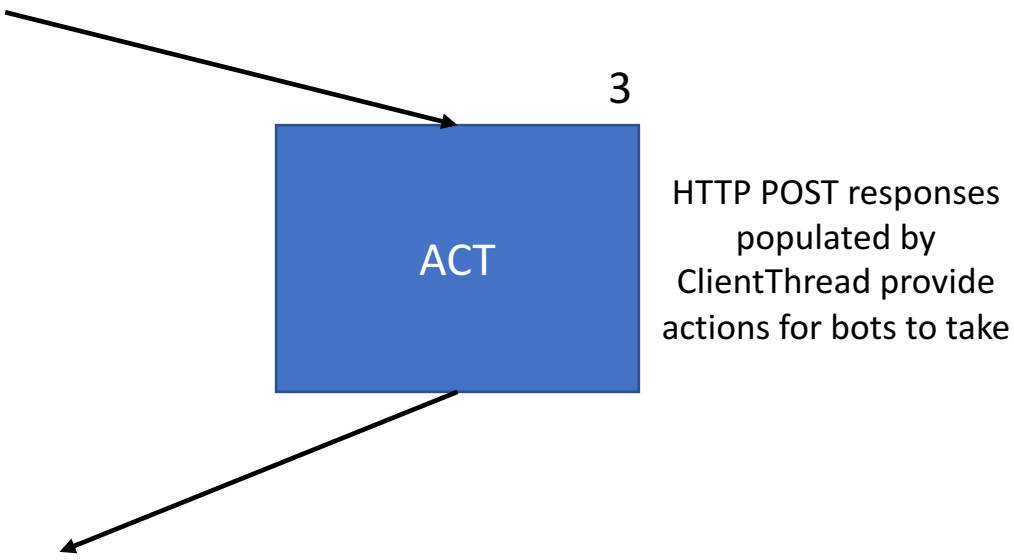
Data Flow:

Component: ACT

Status: COMPLETED

Issues:

- None currently



Component: Bot Code

Status: STARTED

TODOs:

- Need to write **appropriately abstracted** bot “actions” that are hard-coded functions

Issues:

- TBD

In-game bot code takes actions sent
by ACT for each unit & team

Discussion on “Appropriately Abstracted”

- Our RL* agent can learn at various layers of decision making in the game (up to us how much we want to be AI vs hard-coded logic)
  - Example Learned Decisions: { NONE }
    - we hardcode all logic into bot-code
    - the current state of most/all bots in Workshop today
  - Example Learned Decisions: { PUSH LANE *X*, DEFEND LANE *Y* }
    - we hardcode all logic about moving, attacking, ability/item usage, runes, roshan, jungling, item purchase selections, minions/illusions, glyph use, courier use, etc.
    - we learn when to ”push” (go on offense) or “defend” (go on defense) for specific lanes
      - +ve reward when any enemy building (tower, barracks, etc.) is destroyed
      - -ve reward when any friendly building (tower, barracks, etc.) is destroyed
      - Possible world model: location of buildings, health of buildings, location of all units (friendly and enemy), health/mana of all units, combat power of all units

* we can use supervised (and possibly unsupervised) learning for many of these as well