Challenge 1 : To identify player type

Agent model submitted on 18 March 2025 do following steps to identify the player type

After warming up and sending random keys , ‘a’ key will be pressed and rect player position will be checked and if it changed then it will be RECT player otherwise DISC player.

But this will cause problem when two agents starts playing and do the same actions and there will be confusion

Solution : solution to this problem was found by updating act-r-experiment.lisp

Read-player-type function implemented which will check (:playing rect) or (:playing disc) message received from Game server.there was challenge as initially some telnet codes were sent and was not easy to implement this.

```
;; Function to read the player type message
(defun read-player-type (socket)
  "Read the player type message from the socket"
  (let ((buffer (make-array 512 :element-type '(unsigned-byte 8) :initial-element 0)))
    (handler-case
        (progn
          ;; Read the player type message from the socket
          (let ((bytes-read (sb-bsd-sockets:socket-receive socket buffer nil)))
            (when bytes-read
              ;; Find the actual message length (look for CR+LF)
              (let ((message-end (or (position 10 buffer :start 1 :from-end nil) bytes-read)))
                ;; Convert the bytes to a string (excluding any trailing nulls)
                (let ((message (sb-ext:octets-to-string
                                (subseq buffer 0 message-end)
                                :external-format :utf-8)))
                  (format t "Received message: ~A~%" message)
                  ;; Extract player type
                  (let ((data (ignore-errors (read-from-string message))))
                    (when (and (listp data) (eq (car data) :playing))
                      (setf *player-type* (cadr data))
                      (format t "We are controlling the ~A player~%" *player-type*)))))))))
      (error (e)
        (format t "Error reading player type: ~A~%" e)))))
```

```
;; Function to skip the telnet protocol bytes
(defun skip-telnet-bytes (socket)
  "Skip the telnet control bytes on the socket"
  (format t "Skipping telnet control bytes...~%")
  (let ((buffer (make-array 6 :element-type '(unsigned-byte 8))))
    (handler-case
        (progn
          ;; Read 6 bytes directly from the socket
          (sb-bsd-sockets:socket-receive socket buffer nil)
          (format t "Skipped telnet bytes: ~A~%" buffer))
      (error (e)
        (format t "Error skipping telnet bytes: ~A~%" e)))))
```

Player info will be continuously update in visicon so that in model we can read it

```lisp
(defun respond-to-key-press (model key)
  "forward key presses to game server and update visual buffer"
  (declare (optimize (safety 3) (debug 3)) (ignore model))
  ;; send data
  (ensure-connection)
  (when *gstream*
    (clear-input *gstream*)
    (format *gstream* key)
    (finish-output *gstream*)
    ;; read updated scene
    (multiple-value-bind (updated-scene err) (ignore-errors (read *gstream* nil nil nil))
      (when err
        (format *error-output* "~&error reading from game server (err: ~w).~%" err))
      (when (consp updated-scene)
        (format *standard-output* "~&installing scene in visicon, scene: ~w~%" updated-scene)

        (delete-all-visicon-features) ; reset visicon

        ;; Add player-type information to visicon first so it's always available
        (when *player-type*
          (add-visicon-features `(isa (text-feature text)
                                      screen-x 5
                                      screen-y 5
                                      value (text "player-info")
                                      text ,(format nil "Player Type: ~a" *player-type*))))
```

Working code for this can be found on the following location

https://github.com/SandeshGavhane/DeepSeek_ICA_Agent/blob/main/geomates/DeepSeekAgent_Version3_Reactive_PlayerDetection.lisp

https://github.com/SandeshGavhane/DeepSeek_ICA_Agent/blob/main/geomates/act-r-experiment.lisp