filed on: {indigo}<kbvlsi>loopscourse>truckinmanual.bravo
Last Edited: SM: July 11, 1983

Truckin MANUAL

by the Loops Design Team

Daniel Bobrow, Sanjay Mittal, and Mark Stefik

This document gives the basic instructions for creating game boards, starting, stopping, and continuing a game, interrupting a game in the middle, and attaching gauges to monitor the internal state of *Truckin* players.

[NB: *Truckin* now has versions which run on both single machines as well as multiple-machine configurations. The following instructions are written for the single machine version. Any differences for the multi-machine version are indicated in smaller print. Otherwise the instructions apply to both versions.]

A. Creating a new game

Send the message *New* to *$Truckin* as follows:

(← $Truckin New)

this creates a new game board and the lisp variable *PlayerInterface* is set to the instance of *TruckinPlayerInterface*. All commands sent by your player or you go to ***PlayerInterface***. You can play any number of times on this basic game board as follows.

[On a *RemoteMasterMachine*:
(← $MasterTruckin New) creates a new game.
On a *RemoteSlaveMachine*:
(← $SlaveTruckin New) sets up the Truckin world and links your machine to the MasterMachine. You will be asked for a *unique name* to identify your machine and the address of the *PostMaster*. Please ask the game coordinator for this address. A new game cannot be created from a slave machine - the slave machine will run the game created by the master machine.
PlayerInterface is set as above in both these cases as well].

[[[[In all these cases, upto 4 arguments can be given to the New message to select the game configuration you want. The description above is for the default case.
Arg 1: Type of Game--
This specifies what kind of *DecisionMaker and PlayerInterface* you want. Currently the only value is **TimeTruckinDM** (the appropriate player interface is automatically selected). Later, we may put in other versions of the game.
Arg 2: Type of Game Board--
This specifies what kind of game board you want: **BWTruckin** or **ColorTruckin**. The former is the default. In order to use ColorTruckin option, you need a color monitor attached to your machine.
Arg 3: Type of Simulator--
This specifies whether you want the game board to be displayed or not: **DisplayTruckinS** or **NoDisplayTruckinS**. The former is the default. The NoDisplay version of the simulator maintains an upto date version of the game but does not display the game board.
Arg 4: Broadcast List--
This is a list of objects who want to receive a copy of all game messages which change the world. These objects must be capable of responding to the messages described in the *MultiMachineTruckin* document. These objects will get the messages after the world has already been updated.]]]]

B. Starting a game

Send the message *BeginGame* to *PlayerInterface* to start a game as follows:

(← PlayerInterface BeginGame)

This message refreshes the game board created earlier and prompts you for the players you want in this game. You can either create new players from among the existing player classes (via an interactive menu) or use any players created earlier. [The menu appears next to the prompt window at the left top of the screen]. The menu for the players offers you a choice of both player classes and existing player names. You can opt for all existing players by choosing the *ALL-EXISTING* menu option. Select *NO* when you are done selecting players.

You can pass one optional arguments in the BeginGame message.

Arg : If T then all existing players will be used for the game and you will not be asked for players. This might be convenient during debugging when you want to use the same game board and same set of players for debugging your player.

[[If you are running *SlaveTruckin*, BeginGame will let you select your local players, but the game will only start when the Master Machine decides - which it does when a BeginGame is done on the Master Machine.]]

C. Interrupting a game in the middle.

In addition to the rule exec and the break/trace facility of the rule language (see Rule Language manual), there is another way to temporarily stop a game in the middle and bring up the lisp user exec. Hold the CTRL and LEFT SHIFT keys simultaneously when one of the trucks is moving. This will put you into the Lisp User Exec, where you can examine things and/or edit your rule sets and functions. Type **OK** in the Exec to resume the game. On a dorado, the trucks move pretty fast, so if the above does not work the first time, try again.

C.2 Interrupting a player any time

Left of the Status Window, you will notice a menu which lists the players running on your machine. Selecting any player in the menu, allows you to interrupt that player and bring up the Rule Exec. Remember that the game time continues to click while you are in the Rule Exec.

D. Suspension/Premature Termination of the Game

You can suspend, resume, or kill the game by using the **Game Control Menu**, which normally appears left of the Status Window. Selecting *Suspend* will suspend the game (but remember that the time allocated for the game continues to tick, so when you resume, the intervening time will be deducted from the game time). Selecting *Awake* will resume the game and *Kill Game* will kill the game.

E. Attaching gauges to your player's truck

You can attach gauges to *Instance Variables* (IVs) of objects under your control such as your player or truck in order to monitor important internal state during the game. When you first create a player, the game master will offer to put gauges on your truck, i.e., to the IVs *cashBox, fuel, weight,* and *volume*. You have several options. **NO** will not put any gauges. **YES** response will lead to the system asking you whether you want gauges on each of the four IVs listed above. For each IV for which you respond with *YES* the system will offer a choice of gauges. **DEFAULT** response will put default gauges on fuel.

Once you put gauges on a player, they can be reused when you use the same game board for a new game or create new game boards. Thus, if you expect to use a player many times, it pays to attach the desired gauges once and continue to use the player.

F. Attaching gauges to other IVs of your player

When you create a player, the instance object is given the same name as the driver name you enter. Thus, if you name some player **Joe** you can access the object as **$Joe**.

You will often find it useful to attach gauges to IVs of your player. For example, if your player is an instance of **Peddler**, you might want to monitor IVs such as *destination, stoppingPlace,* and *goal.* The way to attach gauges on your player is to send it the **AddGauges** message. For example,

(← $Joe AddGauges '(destination goal)

will attach gauges to destination and goal IVs of $Joe if $Joe is an instance of Peddler. The AddGauges method will prompt you for the type of gauge. The most suitable gauge for arbitrary values is **LCD.**

The AddGauges message can be used to select default gauges on the instance variables indicated, instead of having to select gauges yourself each time. In order to do this, you have to specify additional information in the object class as shown in the following simplified description of the class **Truck.**

```
(DEFCLASS Truck
   (MetaClass ..)
   (Supers ..)
   (ClassVariables ..)
   (InstanceVariables (cashBox 10000 DefaultGauge LCD GaugeLimit (0 10000))
                      (fuel 80 DefaultGauge Dial GaugeLimit (0 80))))
```

Thus, suppose, you wanted an LCD gauge to be the default gauge on *destination,* you can specify this for use by the AddGauges method by adding the property *DefaultGauge* to the instance variable *destination* with *LCD* as the value. Then pass T as the *second argument* in the above AddGauges message. This will result in a LCD gauge being installed on destination and you will be prompted only for goal. [You can do the same for *goal* or any other IVs also]. If the default gauge you have specified is being used for numbers, you also should specify the default limits. For this, put under the *GaugeLimit* property a list containing the two numbers which indicate the lower and upper limits.

G. Adding gauges under program control

You can also attach gauges under full program control by specializing the method **SetUpGauges** in the class **Player.** The description given above is carried out by this method. You could write your own SetUpGauges method in your player class and make it attach gauges by using the method *AddGauges* described earlier. Both **Truck** and **Player** respond to the message AddGauges. This way you could build into your SetUpGauges method, your choice of gauges, which then will be carried out by the system each time you create a player of that class.

H. Selecting trucks under program control

You can also select the truck you want for your player automatically, instead of being prompted for it. In order to do this specialize the method **SelectTruck** for your player class. This method will be called when your player class is instantiated. This method should return the name of one of the truck classes currently allowed in the game. Currently, the allowed trucks are: **MacTruck, GMCTruck, FordTruck,** and **PeterBiltTruck.**

I. Summarizing the truck data at a glance

You can get a summary report of your players truck by sending your player (say **Joe**) the **Show** message as follows:

(← $Joe Show)

This will print out the cashBox, fuel, weight, and volume, as well show you the cargo your truck is carrying. This summary may be useful during debugging.

J. Clearing up the screen

If your screen gets messed up for some reason, you can restore it to the initial state by buttoning the LoopsLogo in the middle top of your screen and selecting the command *SetUpScreen.* You can also do this in the middle of the game when you are in any of the rule exec, user exec or break exec. Even though the game board and gauges will disappear temporarily, they will come back as those windows are written to.

K. When players get control

A player gets control when his/her turn comes and the game master sends a **TakeTurn** message to the instance of your player object. Your top-level rule-set must be written to respond to this message.

You can also write your player in such a way that the top-level rule set never returns, i.e., the *TakeTurn* rule-set uses *whileAll* control structure. The *PlayerInterface* will suspend you when you make a *Buy, Move,* or *Sell* request and reschedule you when your turn comes again.

L. Legal requests by players during game

A player can make three kinds of requests during the game: **Move, Buy, Sell.** After each request, the player is suspended until the request is completed and your turn comes again (i.e., all other players have used up the same amount of time).

1. (← PlayerInterface Move player numOrLoc)
This is a request to move *player* from the current location to a location determined by *numOrLoc.* If numOrLoc is a number, then it is the relative offset from the current location. It can be positive or negative. It can also be the actual instance object representing the particular roadStop in the game.

2. **(← PlayerInterface Buy player qty)**
This is request to buy *qty* of the commodity at the location at which *player* is currently parked.

3. **(← PlayerInterface Sell player commodityInstance qty)**
This is a request to sell sell *qty* of the commodity *commodityInstance* owned by the player in their truck's cargo, at their current location. If qty is not specified, then the qty in the commodityInstance will be used.

The standard value of *player* in all the three above messages is *self* which is bound to the player executing the rule-set.