**Behavior Replicator**

The main ideea of this part our autonomus ideea is the following: it is a bit hard to make the robot do all by itself: find capstones, prioritize stones, work with another robot. So, a solution would be to be able to teach the robot what to do on the spot. For this we would need two things: an encoder state of the robot, in which the driver tells him how to move and what to do, and a decoder state of the robot, in whch the robot does what the driver taught him based on the data that he harvested in an external file(to be kept between instances).

**In**

parser for an input .csv file

we needed to be able to read from an external file for the behavior replicator, so we wrote a class based on (<https://github.com/StPaulAcademy/HOMAR-FTC-Library/blob/master/src/main/java/edu/spa/ftclib/internal/Log.java>) , which parses a csv file

constructor: creates an In class instance

-parameters:

* String directoryPath: the path to the directory in which the input file should be located
* String filename:a string with the desired name of the file from which we read

-methods:

* isDisabled(): returns a boolean value equal to the state of the object(true if disabled, false otherwise)
* setDisabled(boolean disabled): sets the state of the current object to the parameter one
* close(): closes the input parser
* update():reads the next line from the file
* getSData(): extracts the first field of the current line
* getByte(): returns the first field of the current line converted to Byte
* getBoolean(): returns the first field of the current line converted to boolean
* getShort(): returns the first field of the current line converted to short
* getInt(): returns the first field of the current line converted to int
* getLong(): returns the first field of the current line converted to long
* getFloat(): returns the first field of the current line converted to float
* getDouble(): returns the first field of the current line converted to double

**Log**

writer to an output .csv file

we needed to be able to write to an external file for the behavior replicator, so we wrote a class based on (<https://github.com/StPaulAcademy/HOMAR-FTC-Library/blob/master/src/main/java/edu/spa/ftclib/internal/Log.java>) , which parses a csv file

constructor: creates an Log class instance

-parameters:

* String directoryPath: the path to the directory in which the output file should be located
* String filename:a string with the desired name of the file to which we write

-methods:

* isDisabled(): returns a boolean value equal to the state of the object(true if disabled, false otherwise)
* setDisabled(boolean disabled): sets the state of the current object to the parameter one
* close(): closes the output parser
* update(): writes the current line to the file
* addData(T x):adds the string variant of the parameter x to the current output line. T can be String, Object, byte, boolean, char, short, int, long, float, double

**SaveState**

we needed a class in which we could store input states for our behavior replicator, so we created a base class to be inherited by it’s two derived subclasses, which could store all the inputs we could possibly need. It also contains the name of the file and the directory path to the save states. This is necessary to assure that the input file coincides with the output file through inheritance.

(constructor): because this class is supposed to be inherited, the constructor just initializes every tracked input with a neutral value

**InputSaveState**

this is the derived class of SaveState which is designed to read save states from a file. We need this so we can simulate driver and sensor inputs on the decoder end of our behavior replicator

constructor: initializes everything with a neutral value, same as in the base class

methods:

* init\_in(): initializes the input parser
* is\_invalid(): return a boolean value which tells us if we reached EOF
* read(): reads all the data we need from input file, in the same order in which it is written by an OutputSaveState instance

**OutputSaveState**

this is the derived class of SaveState which is designed write save states to a file. We need this so we can record the driver’s instructions via our encode end and afterwards feed them to the decoder end of our behavior replicator

constructor: records the current state of our sensor and driver inputs

* parameters: EncodeController controller: needed to be able to get driver inputs
* change(OutputSaveState other):this copies another’s OutputSaveState fields to the current OutputSaveState
* init\_out(): initializes the output writer
* write(): writes all the data we need to an output file, in the same order it is requested by InputSaveState
* write\_final():writes one last time the data an then closes the filename
* is\_same(OutputSaveState other): checks if two OutputSaveStates are the same, differing only maybe by time
* update(OutputSaveState other): updates the time of the current save state with the other one’s time

**GeneralController**

we needed a class to act like “a man in the middle”, so we could easly copy all the inputs that this class is provided with, and be able to easly pretend to be getting inputs, but actually replicating them from a save state. This class is made to work with the default drivetrain, but through inheritance it can help both our encoder and decoder ends. We also keep an ElapsedTime object to be able to keyframe the SaveStates

constructor:gets a refference to the current opmode and sets up hardware sensors(if any)

-parameters:

* LinearOpMode opmode: needed to get the gamepad inputs

methods:

* get\_left\_stick\_y(): returns the value of a gamepad’s left\_stick\_y value
* get\_left\_stick\_x(): returns the value of a gamepad’s left\_stick\_x value
* get\_right\_stick\_y(): returns the value of a gamepad’s right\_stick\_y value
* get\_right\_stick\_x(): returns the value of a gamepad’s right\_stick\_x value
* get\_x(): returns the value of a gamepad’s x value
* get\_y(): returns the value of a gamepad’s y value
* get\_a(): returns the value of a gamepad’s a value
* get\_b(): returns the value of a gamepad’s b value
* reset\_runtime():resets the runtime to avoid having to remember the starting time of it
* get\_dpad\_up(): returns the value of a gamepad’s dpad\_up value
* get\_dpad\_down(): returns the value of a gamepad’s dpad\_down value
* get\_dpad\_left(): returns the value of a gamepad’s dpad\_left value
* get\_dpad\_right(): returns the value of a gamepad’s dpad\_right value
* get\_left\_bumper(): returns the value of a gamepad’s left\_bumper value
* get\_left\_trigger(): returns the value of a gamepad’s left\_trigger value
* get\_right\_bumper(): returns the value of a gamepad’s right\_bumper value
* get\_right\_trigger(): returns the value of a gamepad’s right\_trigger value
* getDistance():returns the value of the distance sensor’s distance
* getDirection():returns the value of the compass sensor’s direction
* red():returns the value of the color sensor’s red value
* green():returns the value of the color sensor’s green value
* blue():returns the value of the color sensor’s blue value
* get\_runtime(): returns the current runtime

**EncodeController**

inherited derived class responsible for our encode end of the behavior replicator

constructor: same as base class constructor, with the addition of initializing an OutputSaveState for logging data

-parameters

* LinearOpMode opmode: needed for recording gamepad inputs

-methods

* write\_state(): writes the current OutputSaveState if necessary
* write\_final\_state():writes the current OutputSaveState and then closes the output writer

**DecodeController**

inherited derived class responsible for our decode end of the behavior replicator

constructor: same as base class constructor, with the addition of initializing an InputSaveState for reading data

-parameters

* LinearOpMode opmode: needed for recording gamepad inputs

-methods

* reads\_state(): reads the next InputSaveState if the current one is outdated
* is\_invalid():return true if EOF was reached
* get\_left\_stick\_y(): returns the value of a SaveState’s left\_stick\_y value
* get\_left\_stick\_x(): returns the value of a SaveState’s left\_stick\_x value
* get\_right\_stick\_y(): returns the value of a SaveState’s right\_stick\_y value
* get\_right\_stick\_x(): returns the value of a SaveState’s right\_stick\_x value
* get\_x(): returns the value of a SaveState’s x value
* get\_y(): returns the value of a SaveState’s y value
* get\_a(): returns the value of a SaveState’s a value
* get\_b(): returns the value of a SaveState’s b value
* get\_dpad\_up(): returns the value of a SaveState’s dpad\_up value
* get\_dpad\_down(): returns the value of a SaveState’s dpad\_down value
* get\_dpad\_left(): returns the value of a SaveState’s dpad\_left value
* get\_dpad\_right(): returns the value of a SaveState’s dpad\_right value
* get\_left\_bumper(): returns the value of a SaveState’s left\_bumper value
* get\_left\_trigger(): returns the value of a SaveState’s left\_trigger value
* get\_right\_bumper(): returns the value of a SaveState’s right\_bumper value
* get\_right\_trigger(): returns the value of a SaveState’s right\_trigger value
* getDistance():returns the value of the distance sensor’s distance
* getDirection():returns the value of the compass sensor’s direction
* red():returns the value of the color sensor’s red value
* green():returns the value of the color sensor’s green value
* blue():returns the value of the color sensor’s blue value