## Interpreter Component

Within the game system, the Interpreter component is responsible for parsing the programs of AI robots. These robots are files externally created and encoded as JSON files. Ideally, the system would draw these from an online collection of robot records, but the software as deployed can only retrieve robots which are stored locally, in the /resources/ExampleRobots folder. Some of these robots were discovered to not follow the specifications for the robot language, and only the robot 0FixedCentralizer.jsn has been confirmed to work properly. However, the Interpreter should be capable of parsing and playing any correctly formatted robot script.

Each AI program is loaded as a list of strings representing each line of the program, and when an AI player is initialized, these strings are stripped of their comments, whitespace removed, and then appended to a clean list as sanitized program data. When a match begins, the pieces for each robot player are initialized with their respective programs. These programs are run from start to finish in an initialization mode, where certain built-in functions of the parser (such as moving a robot or shooting) are disabled. This has the effect of creating and populating any program variables and defining new functions (“words”). Each robot program requires a play word to be defined to be run properly; if a program does not define one, the Interpreter creates a blank one.

Every operation performed by the Interpreter on behalf of a program involves either the list of current instructions, or a stack used for operations. When a piece’s turn is called, the play word is placed on the list of current instructions, and the parser called. The parser replaces the play word with its code (defined during initialization), and then the parser will continue to run until the list of current instructions is empty, upon which the Interpreter will end the piece’s turn.

The parse() method is the core of the Interpreter. It examines each term of the list of current instructions and performs the appropriate action on it after determining its nature.

* If the term is an empty string, the parser will do nothing.
* If the term is a semicolon, the parser will clear all loop iterators and reset the semicolon flag.
* If the term is “I”, the parser will attempt to retrieve the value of the current loop’s iterator and place it on the stack, ending the AI’s turn and throwing a debug error message if no iterator exists.
* Then parser searches the Interpreter’s list of standard built-in functions (43 in all, including arithmetic operations and board queries) to see if the term matches any of these. If so, it will call the built-in function using the Interpreter’s function table. Regardless of whether the called operation succeeds or fails, the parser returns normally. If the mode is illegal (trying to call restricted operations like shoot() during initialization), it will throw an error message, clear the current instruction list and exit.
* Next, the parser searches the list of user-defined words for this program to see if the term matches. If it finds a match, it appends the replacement instructions for that word to the beginning of the current instruction list.
* The parser then searches the list of user-defined variables for this program to see if the term matches. If it finds a match, it pushes the “address” of the variable to the stack by appending a # symbol to the term and pushing it to the stack.
* If none of the above are true, the term is a value that should be pushed to the stack. The parser checks if this term begins with a “.” Character, signifying a string, and if it is a string, it strips out the dot and quotation marks before pushing the value to the stack.

In order to protect the user against the adverse effects of malformed or malicious user-created programs, the Interpreter was created with a no-tolerance policy for failing code. Since many operations involve the stack, if one operation has incorrect parameters, it is assumed that it will not leave the stack in an appropriate condition for the following operation, having a domino effect on all operations afterwards. Therefore, if an operation fails for not having correct arguments or if some other error state occurs, the Interpreter immediately clears the current instruction list and the stack, which has the effect of stopping the AI’s turn after that operation.

Additionally, the Interpreter runs a timer on any loop-related operations, in order to prevent infinite loops from locking up the software or causing delays. This timer is set at the beginning of loop operations, and if the timer expires, the AI’s turn ends immediately.