"The practitioner of literate programming can be regarded as an essayist, whose main concern is with exposition and excellence of style."
—Donald Knuth, “Literate Programming”
Despite the limitations of the AGC code, which range from the concise syntax of Yul and AGC's extended yet still restricted interpretive language, the small amount of punch card space available for use in coding each line, and the requirement of a compact body of instructions, the AGC code contains a wealth of imaginative and highly creative language. Some of the wordplay found in the AGC code resembles what John A. Barry calls "technobabble": specific technological metaphors invented to describe and explain computing that are also frequently used to describe human behaviors in computing terms. But there are also references and riffs on popular culture, contemporary political events, and other textual sources. Perhaps in acknowledgment of the poverty of the system's vocabulary, the AGC programmers filled the code with rich, descriptive language and cultural and literary references. We can find playful use of language from the most visible elements of the system, the main astronaut interface to the computer, to one of the most hidden components, the use of extended literary quotes within the remarks section of the code that would never even be processed and interpreted by the assembling programs.

One of the major innovations of the Apollo Guidance Computer project was the development of what was essentially real-time computing. In a 2001 group interview with other MIT Instrumentation Lab AGC programmers, Albrecht "Alex" Kosmala remarked on the incredulity he experiences when he explains to programmers that this "real-time control computer with an event-driven, asynchronous executive" was developed in the 1960s. The AGC needed to quickly respond to input from instruments and the astronaut interface. Rather than operating in a synchronous fashion, meaning the linear completion of one computational task after another, the programmers wanted to design a system that could very quickly shift from the execution of one job to another. The implementation of this system required the addition of a significant amount of code but it made possible both the asynchronous execution and a powerful form of error recovery. The "executive" referenced by Kosmala is a distinct program and the major enabling technology for these advanced functions. The EXECUTIVE program's respon-
sibility is to run jobs and to make sure that it is always running the job with the highest priority.

The EXECUTIVE combined with another program called WAITLIST, which managed the queue of smaller bits of code called tasks, provided the AGC with the ability to interrupt and resume execution of programs. Together WAITLIST and EXECUTIVE provided essentially what we call an operating system for the AGC. The WAITLIST program is dated in the code as being written on October 10, 1966 and modified four times. The following is slightly reformatted code from the scanned images:

```
L WAITLIST
R0001 PROGRAM DESCRIPTION       DATE -- 10 OCTOBER 1966
R0003 MOD NO -- 2       LOG SECTION -- WAITLIST
R0005 MOD BY -- MILLER    (DTMAX INCREASED TO 162.5 SEC)
                    ASSEMBLY -- SUNBURST REV 5
R00072 MOD 3 BY KERNAN (INHINT INSERTED AT WAITLIST) 2/28/68 SKIPPER REV 4
R00073 MOD 4 BY KERNAN (TWIDDLE IN 54) 3/28/68 SKIPPER REV 13.
R000799
```

This code listing shows that there were major and minor modifications of the code as a whole and that these major modifications involved a renumber. The modification that introduced MOD 3 on February 28, 1968 was most likely part of the same major modification as MOD 4 on March 28, 1968, as the added cards (R00072 and R00073) were not renumbered. MOD 3 introduced inhibited interrupt (INHINT) mode during WAITLIST, ensuring that tasks run through the WAITLIST were executed to completion. WAITLIST was used, as the code reads, TO CALL A PROGRAM (CALLED A TASK). It had a limited list of nine tasks that could be in the queue or task list. If the length of tasks exceeded the maximum number, the program executes (in other words, TCF or transfers control) the following routine, called WTABORT:
WTABORT TC FILLED
NOOP # CAN'T GET HERE
AD ONE
TC WTLST2
OCT 10

... 
FILLED DXCH WAITEXIT
TC BAILOUT1 # NO ROOM IN THE INN
OCT 01203

WTABORT, again, executes another instruction, this one called FILLED, that eventually “bails” out of the program and produces an alarm state. The code in this section is concerned with error states and how to address the problem of running out of space, of not having any rooms in the inn. There are checks to make sure that there is truly a “no vacancy” or FILLED state before generating an alarm. The WAITLIST “tasks” were required to be short running – they could run from 0.01 seconds (a centisecond) to 162.5 seconds – and thus much of the code for this program concerns waiting, counting time, and the timing of the tasks.

The AGC, however, did have some basic interrupt and restart features in the form of restart protection prior to the addition of the EXECUTIVE. Restart protection depended upon the existence of restart points stored in erasable memory that could survive power loss. These restart or checkpoints were scattered throughout the code at crucial points. When the AGC was restarted, it resumed execution at the restarted point, restoring the previous saved state. Restarts could be forced by software – a feature part of many contemporary operating systems known as a kernel panic – in order to protect from overloading and potential corruption of data. We can see an example of this in Don Eyles's explanation of the origin of an oddly named instruction known as WHIMPER that appears as such in the Apollo 11 AGC code:
In a previous version of the code, Eyles explains, WHIMPER appeared in a simple form with a playful and somewhat helpful remark: “The instruction at tag WHIMPER transferred control to the instruction at TAG WHIMPER, whereupon TC Trap would detect the endless loop and trigger the restart.” The line, according to Eyles, appeared as such prior to the Luminary 1A build 099 that was used in the Apollo 11 flight:

```
WHIMPER TC WHIMPER NOT WITH A BANG....
```

The remark included in this earlier code quotes, within the designated remark space of the card, part of the final line of T.S. Eliot’s 1925 poem “The Hollow Men,” which was preceded by “This is the way the world ends.” The AGC programmers allude to these lines in their naming of the condition of a forced restart by the TC Trap job. The TC Trap job monitored the progress of other programs and was capable of initiating a software (rather than hardware) restart to reestablish a known state for the AGC. The programmers explicitly referred to this procedure – an innovation in computer systems design – as a software restart. Calling the WHIMPER instruction triggers an endlessly looping recursive state which would necessitate a restart – the end of the world for the computer, but much better than a possible “bang” as a result of an unrecoverable and undetected error or an inoperable AGC. Following the revision of the WHIMPER function to no longer produce the self-referential trap condition and the removal of the no-longer relevant remark, the name stuck “for sentimental reasons” without any markers to indicate the original referent. WHIMPER remained in the code as the name of the subroutine used as part of a software restart.

---

Far less disruptive to the operation of AGC than a TC Trap software-initiated restart was the normal process of interruption used by the EXECUTIVE program. The AGC programmer’s manual describes the process of interruption:

This means that the normal sequence of instructions of a program may be broken into at any point, and that control is transferred to some other program. There is a short subroutine which has the net effect of returning control to the original (interrupted) program, with no loss of information if certain precautions are taken.4

The occurrence of an interrupt signal stops or breaks the execution of currently running programs to run a program with a higher priority. These occur frequently and are what makes real-time computation possible. The notion of an interrupt has found widespread adoption in computing to enable multitasking, quick responses to events and triggers, and to address the common problem of input devices running at much slower speeds than processors. The EXECUTIVE program continuously runs an idling program or subroutine known as DUMMYJOB. This subroutine has the lowest priority, thus making sure that will only be executed when nothing else needs to be computed.

```
DUMMYJOB    CS    ZERO     # SET NEWJOB TO -0 FOR IDLING.
    TS    NEWJOB
    RELINT
    CS    TWO     # TURN OFF THE ACTIVITY LIGHT.
    EXTEND
    WAND   DSALMOUT
```

The DUMMYJOB instruction is introduced with a remark that explains that the idling process “is not a job in itself, but rather a subroutine of the executive.” If there are no jobs running, the DUMMYJOB subroutine is executed, turning off the activity light and making sure that the EXECUTIVE is available for running any jobs. In turning off the activity light, the AGC alerts the astronaut that the system is idling, running DUMMYJOB. In the above code, we

see the instruction RELINT called. This instruction enables interrupts and is generally used in combination with INHINT, an instruction that inhibits interrupt activity for brief and important tasks that cannot be safely interrupted. The instruction TC RIP is used to Transfer Control to Resume Interrupted Program, which restarts the preserved state of the prior program.

Military-style acronyms fill the world of the Apollo mission and many make their way into the code. Short and concise, these acronyms compress language into the smallest amount of space required to communicate a concept. Acronyms were also particularly well suited to the computational environment because these computers and the devices used to input and store the code, the punch card systems, all had limited space and required the use of capital letters. Much of the wordplay appearing in the code turns on the ambiguities and slipperiness of these otherwise precise, short terms. The Lunar Module was abbreviated everywhere in the code as LM, which was always pronounced as “Lim.” The main input device for operating the guidance computer was called the “Display and Keyboard.” This name was shortened to DSKY, which the programmers and astronauts pronounced as “Dis-key,” which Don Eyles explains was pronounced to rhyme with whiskey.5

The program that was responsible for handling the DSKY user interface—in other words, responding to the astronaut’s manual input and displaying output values, alarms, and present system status—was playfully named PINBALL GAME BUTTONS AND LIGHTS. The DSKY was operated by entering a two-digit “verb” to perform an action on another two-digit object or “noun.” A set of remarks in the code explains the logic behind this mode of communication with the computer:

```
# THE LANGUAGE OF COMMUNICATION WITH THE PROGRAM IS A PAIR OF WORDS
# KNOWN AS VERB AND NOUN. EACH OF THESE IS REPRESENTED BY A 2 CHARACTER
# DECIMAL NUMBER. THE VERB CODE INDICATES WHAT ACTION IS TO BE TAKEN, THE
# NOUN CODE INDICATES TO WHAT THIS ACTION IS APPLIED. NOUNS USUALLY
# REFER TO A GROUP OF ERASABLE REGISTERS.
```

On the far left side of the DSKY were the VERB and NOUN buttons. On the far right, ENTR and RSET. To operate the DSKY, to perform some task, the astronaut pressed VERB and then entered the two-digit program number, then pressed

---

5 Eyles, *Sunburst and Luminary*, 47.
NOUN, followed by the two-digit code for the action, and finally pressed enter or ENTR. The use of a NOUN was not always required; some programs would be run with just a VERB. **Figure 5** shows an image of the summary card with the NOUN and VERB list used for a subsequent version of the AGC. The language of nouns and verbs, one of the most simple yet flexible user interfaces that one can imagine, can be found throughout the code. This is a compact but powerful method of interaction that continues to present in the inverted form of selection (noun) and clicking (noun) in graphical user interfaces.

Despite its functional design imperative and the limited syntax of the Yul programming language, we find a real sense of humor and play within the AGC code. Perhaps this is because all authors and programmers, when exploiting the few given freedoms available in any discourse, have a tendency toward pushing the limits of the existing language. Programming is as much of a “language game” as any other language and the tightened boundaries of a small syntax give way to a sense of play found within any constrained environment. The use of the terms “nouns” and “verbs” within both the hardware and software systems of the AGC invite a playful reading of the code as self-aware of the limits of this particular language. The stripped-down syntax invites exploration of the combination of two-digit noun and verb codes. In a section of code just below the above, in which programmers or “the authors,” as they called themselves, supply lines spoken by Jack Cade in Shakespeare’s *King Henry XI*.6

```
# THE FOLLOWING QUOTATION IS PROVIDED THROUGH THE COURTESY OF THE AUTHORS.
#
# “IT WILL BE PROVED TO THY FACE THAT THOU HAST MEN ABOUT THEE THAT
# USUALLY TALK OF A NOUN AND A VERB, AND SUCH ABOMINABLE WORDS AS NO
# CHRISTIAN EAR CAN ENDURE TO HEAR.”
#
# HENRY 6, ACT 2, SCENE 4
```

These remarks, the only quoted lines but not the only reference to Shakespeare found in the code, add some humor to the programmer’s reliance on the noun and verb structure.7 The two-digit VERB and NOUN thus provided the

6 Ronald S. Burkey has added a note to the code uploaded to Github with the correct citation for this passage: *King Henry VI*, Part 2, Act IV, Scene VII.
7 Hugh Blair-Smith provides his interpretation of these lines in “Annotations to Eldon Hall’s *Journey to the Moon,*” *Apollo Guidance Computer History Project*, February 1997.
Figure 5. Apollo 17 Verb and Noun List.
programmers with the basic naming structure for many of the various programs contained within the body of the AGC code that required interaction with the astronaut. Many of the major programs run by the AGC EXECUTIVE are numbered using two-digit codes.

Don Eyles contributed to the AGC code two routines formally named R11 and R13 that he informally called ROSENCRANTZ and GUILDENSTERN. These “names from Hamlet,” he writes in his memoir, “swam into my consciousness because Tom Stoppard’s Rosencrantz and Guildenstern Are Dead was then playing on Broadway.” 8 The code introduces the R13 routine with three remarks cards, here transformed and remediated into contemporary Github-friendly formatted code:

```
#*******************************************************************
# GUILDENSTERN: AUTO-MODES MONITOR (R13)
#*******************************************************************
```

The remarks continue:

```
HERE IS THE PHILOSOPHY OF GUILDENSTERN: ON EVERY APPEARANCE OR DISAPPEARANCE OF THE MANUAL THROTTLE DISCRETE TO SELECT P67 OR P66 RESPECTIVELY: ON EVERY APPEARANCE OF THE ATTITUDE-HOLD DISCRETE TO SELECT P66 UNLESS THE CURRENT PROGRAM IS P67 IN WHICH CASE THERE IS NO CHANGE.
```

These two routines, as Eyles explains, monitored switches and buttons; GUILDENSTERN, which was split into two lines as GUILDEN/STERN, the switch to manual mode and ROSENCRANTZ, buttons to abort landing. 9 The GUILDRET routine, riffing on GUILDENSTERN, was also added to this section of code. Eyles’s comment addressing the way in which these two names “swam” into his consciousness demonstrates how the presence of natural language and arbitrarily named routines and programs links culture to code. The “philosophy of Guildenstern” became embedded within the code and, fittingly for Eyles’s dramatic reference, this philosophy was attached to buttons that

8 Eyles, Sunburst and Luminary, 105.
9 There are no references to ROSENCRANTZ remaining within the Luminary099 code used in the Apollo 11 flight.
functioned to remove control from the computer and return it to a human, who might know better.