To: Founding Chancellor of Codemanistan Technologies LLC
From: David Reed and Brendan Rejevich, Software Developers
Subject: Go Language and Ecosystem

For these past assignments, Golang has worked well for the needs of our team. Its standard libraries include all of the basic functionality that our team needed to implement (TCP connectivity, JSON encoding and decoding, dynamic code loading*) without external library inclusions. In addition, the libraries that we did use provided simple APIs across multiple data formats, meaning that we were able to develop the translation layer (to and from JSON) in our system independently from the internal components.

The JSON library and the TCP library that we used both connected easily through a single encoder/decoder pair, allowing us to abstract communication over a generic Read/Write interface from Go's io package. On the topic of packages, Go's package structure allowed for easy abstraction barriers between different components, and in fact we could have implemented more abstraction than is currently in the system (abstraction barriers b/w Boards/Workers, Boards/Tiles, etc.), though unnecessary.

While Go can handle dynamic code loading gracefully, our team had trouble getting our Player components to compile within a Linux environment (despite Go gracefully handling cross-OS compilation otherwise). Likewise, Go's major selling point of Goroutines in place of threads turned out to be a problem in the face of infinitely-looping code: the thread safety that Goroutines provide compromises their systems utility, as a user cannot kill a Goroutine externally through any (feasible, reasonable, good) means.

For design docs, Go provides a syntax that allows for the automatic generation of documentation from package-, structure/interface-, and method-level comments. While we wrote comments for the bulk of our structures, we did not end up relying on the godocs for our README, and instead wrote close-approximation sentences describing components.

For our IDE choice, our team was divided between GoLang and Atom, however the functionality between the two was consistent due to how well-developed Go's toolchain is. The different Go tools used (gofmt for formatting, goimports for package management, golint, go build, etc.) were the same for both of us, and we were able to debug in a C-like environment with breakpoints and a stepping tool for fine-tuned bug tracing.