**3.7 Integration And Implementation Reflection**

As described previously, the finished Go problem solver is made up of a series of components - namely the GameEngine, GUI, AI and heuristics. Each of these components had to be integrated together and given communication abilities to create the finalised program. Once integration had occurred, design and implementation evaluation was carried out, allowing insight into any changes that would have been made given more preparation time.

Before the implementation of heuristics, the GUI, AI and GameEngine were created separately and then integrated together. The GameEngine easily interacts with the AI through the use of a private AI variable, allowing the GameEngine to access AI search space, the AI type and AI move selection. The more difficult integration was matching the GUI against the GameEngine and hence allowing it to communicate with the artificial intelligence. Despite the difficulties, this was achieved effectively before heuristics were implemented. The integration was carried out by matching GameEngine methods to GUI buttons and several methods were also added to the GameEngine to allow the GUI to directly interact with the AI instance. After integration, a range of bug fixing occurred and the visual representation of the artificial intelligence within the GUI made it easier to identify problems. Heuristics were then added to the AI later and integration was relatively simple in this respect. *Figure 1* shows the interaction between the finalised components and the success of integration can be seen clearly.

**TBA: ADD FIGURE 1 (Diagram of finished implementation/integration)**

Looking back upon design, implementation and integration there are several design choices and processes that can be evaluated and with more time they may have been different. For example, if this were a long-running Go project several more heuristics would have been design and coded to allow for a more strategic artificial intelligence. Such heuristics include \*ADD HEURISTICS/EXPLANATION\* Also, in terms of board representation the choice of a 2D array was probably not the wisest design choice and a data structure such as a hash map may have allowed for a faster running program and less confusion about the laying out of coordinates within the array. Other design choices that can be reflected upon also include more features within the GUI, such as a more realistic board which could have been created using web-coding, for example HTML [*reference here*], or a different Java package that was not Java 2D [*reference here*]. Another GUI feature that could have been added include allowing the AI to run against itself automatically without being prompted against the user. \*ADD AI EVALUATION HERE\*

The initial wanted outcome of a problem solver allowing users to create life and death problems and play them out against a sophisticated artificial intelligence has certainly been reached through design, implementation and integration. Whilst, some design choices have been reflected upon and evaluated, proving that they could have been better, the end product is still of a high degree and meets the requirement specification.