# 1.0 Proposed Architecture

The game architecture is an architecture of Independent Components (IC). The major components in the architecture are the Logger, the Pieces, the Board, the Display, the Interpreter, and the Flow components. A justification for the architecture will be discussed first, and then the components will be discussed in further detail.

## 1.1 Architecture Justification

The IC architecture was chosen for the game for several reasons, among them being minimal dependency among game entities, the event-driven nature of the game, and the familiarity of the design team with the architecture. Each of these reasons will be discussed in terms of their strengths and risks, and then alternative architectures will be discussed and why they were not chosen.

### 1.1.1 Minimization of Dependency

The most important reason for selecting an IC architecture is that it imposes the minimum dependency among the game entities. This has several benefits. One benefit is that parallelization of class and code construction can be maximized, an important benefit in group-oriented software challenges. Another benefit is modularity of testing and deployment. If the game is effectively decomposed, then testing on a unit level rather than a system level can be maximized, allowing for more controlled and rapid testing procedures.

A risk associated with the minimization of dependency, however, would be the lack of structure. The IC architecture has some of the least structure of any architecture, and as such needs to be carefully managed to prevent issues in later phases. While this risk is noted, the requirements document is quite detailed in the Actor/System interactions, which will provide support here. Like a house with many windows, the detailed nature of the requirements document provides more insight into the game’s inner workings and illuminates the structure.

### 1.1.2 Event-Driven Nature of the Game

Another important reason for selecting IC as the game architecture is that IC architectures effectively works with event-driven systems, and the game is highly event-driven. The IC architecture models events explicitly inside the game system, which makes it easy to use the same framework to think about the system and how it interacts among itself and also with Actors outside the system. That the entire human-game interface relies on events to work properly is a strong secondary line of reasoning here.

The major risk that relying on the IC architecture exposes with event-driven games is clear understanding and elicitation of the events. Should major events not be included in the design, entire communication pathways can be left undersigned, causing problems in the code construction phase. This risk is mitigated for the game partly because of the nature of general game structure. The possible events have been well documented and in these cases, the likelihood of missing major events should be minimal.

### 1.1.3 Familiarity of Design Team with IC

A final important reason for using the IC architecture is that the design team is already very familiar with this architecture. This has the major benefit of allowing the design team to focus its efforts on areas where greater clarity is necessary such as the component analysis and class diagram elucidation. If a less familiar architecture were selected over IC, this would dilute the design team’s concentration and could easily result in a poorer design.

Of course, the risk of familiarity is that an incorrect or subpar choice is made solely due to familiarity. If this occurs, then in the worst case the entire design would have to be redone. However, in this case, it is well-established that games work well with an IC architecture, and this is not the sole or even most important reason for using it.

### 1.1.4 Another Architecture - MVC

An IC architecture is hardly the only applicable architecture that could have been chosen for the game. Another popular architecture is the Model-View-Controller (MVC) architecture. The strength of the MVC architecture relative to the IC architecture is that the assignment of responsibilities is typically much clearer in the MVC architecture. However, this is due to the greater structure provided by the MVC architecture compared to the IC architecture. Should this structure not add value to the design, the design suffers from becoming more rigid than necessary. Another issue with MVC is that the division of testing and code construction responsibilities can be more difficult. Mainly for this latter reason, MVC was not chosen as the primary game architecture. However, as seen below, some components when combined could be seen to provide some of the functionality of the MVC architecture.

//Need to mention: The match supercomponent, how the architecture degrades to MVC outside of matches due to the small number of active components

## 1.2 Component Details

Now that the architecture choice has been justified, the components can be discussed in detail. The components will be discussed primarily in terms of their responsibilities at a large scale, as well as the information flow among the components. This naturally leads to the detailed design discussion in the next section.

### 1.2.1 The Board Component

The central component of the architecture for the actual play of the game is the Board component. It is responsible for maintaining the overall board state and communicating that state to the other components. An example would be to tell the Display when something needs to be updated. Another would be to request information from the Pieces regarding their status (as this is needed to maintain the board state). It is expected that most other components communicate with this component. Looked at from an MVC architectural perspective, the Board has some of the functions of a Controller, but also some aspects of the Model. One of the primary inputs to the Board component comes from the Flow component.

### 1.2.2 The Flow Component

The Flow component holds the responsibility for gathering the Player input—using mouse clicks—and forwarding it to the Board if it had gameplay ramifications, and to the Display if it did not. (//Strictly speaking, this might not be true anymore? Display might need to communicate with Flow very very infrequently?). There is no expectation of communication to the Flow component from any component.

Again, from an MVC perspective, the Event Handler could be partly View in that it interacts with the outside users of the system, but it can also be partly Controller, as it has to know where to send events. The other major View-like component in the system is the Display component.

### 1.2.3 The Display Component

The Display component is responsible for the user interface (UI) elements of the game. It primarily takes information from the Board and displays that information to the user. The Display along with the Flow would be the View component of an MVC architecture. Even more than the Pieces component which appears next, the Display is a passive component and does not //send information out to other components.

### 1.2.4 The Pieces Component

The Pieces component is responsible for storing and tracking information relevant to the pieces of the game. It is not, however, responsible for knowing the location of the pieces; that is handled by the Board component. There is a great deal of communication between the Pieces and the Board. The Board receives the commands from either the Interpreter or the Flow and then adjudicates them based on information obtained from the Pieces component.

//What information does it hold? Examples?

### 1.2.5 The Interpreter Component

As mentioned above, the Interpreter is one of the components that sends information to the Board. As its name suggests, the Interpreter’s main responsibility is to interpret the scripts for the computer player or players and send that information to the Board for processing. It can be seen as the AI analog to the Flow component as it is the primary source of input into the system for computer players in the same way that the Flow component is the primary source of input into the system for human players.

### 1.2.6 The Logger Component

The final component to the architecture is the Logger component. Its responsibility is to store every action that occurs in the game. It also can send that log to the Display when requested along with a given context. For example, if a team asks to view the log, the log displayed should only show the information that that team performed or saw performed.

### 1.2.7 The Complete Architecture

These six components comprise the system architecture. This architecture is summarized in the figure below, along with the important information flows.