# 1.0 Proposed Architecture

The game architecture is an architecture of Independent Components (IC). This architecture works best 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 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.

## 1.2 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.

## 1.3 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.

## 1.4 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.