**Architecture chosen:**

**Layered Architecture**

Components within the layered architecture pattern are organized into horizontal layers, each layer performing a specific role within the application (e.g., presentation logic or business logic).

Reasons for choosing it:

* Reduced dependency because the function of each layer is separate from the other layers and hence forms an abstraction around the work that needs to be done to satisfy the business requirement surrounding the layer:
  + Each layer of the layered architecture pattern plays a specific role and responsibility within the application.
  + Most layered architectures consist of 4 standard layers: presentation, business, persistence and database.
* Testing is easier because of the separated components, each component can be tested individually.
  + This feature may also be known as the separation of concerns between components.
  + Components within a layer only need to handle the logic that pertains to that layer.
  + Effective roles and responsibilities can hence be fitted and built into the architecture, providing ease for the development, testing and maintenance of the overall system.

**Other architectures proposed:**

**Event-driven**

Pros:

* Easily adaptable to complex, often chaotic environments
* Easily extendable when new event types appear

Cons:

* Testing can be complex if the modules affect each other. While unit testing is easy, the interactions between them can only be tested in a fully functioning system
* Error handling can be difficult to structure, especially when several modules must handle the same events

**Microkernel**

Pros:

* Microkernel architecture is small and isolated and can hence function better.
* Security is ensured because only the components that are included will be able to disrupt the functionality of the system.
* Increased stability and security results in decreased complexity of code which runs on the kernel mode.

Cons:

* Deciding what belongs in the microkernel is a difficult process. Choosing the right granularity for the kernel functions is difficult to do in advance, but quite impossible to change in the future
* Modifying the microkernel can be very difficult or even impossible once a number of plug-ins depend on it.
* Context switches and function calls would be required when the drivers are implemented as procedures or processes.

**Client-Server**

Pros:

* Data is centralized within the system and maintained in one place.
* System is hence more secure and serves to increase data security.

Cons:

* Too many users at once can cause latency and congestion.
* Centralization can cause the entire architecture to be disrupted if the primary server goes down.
* Requires highly technical equipment, such as server machines for the maintenance of the network.

As we have chosen to use Unity as the main development environment for our software, we believe that the layered architecture would be most suitable. Apart from the pros and cons of the layered architecture above, the nature of the game, which consists of various functions and interfaces with their individual roles, would best suit a layered architecture. In our case, each layer would serve a specific function. For example, the GUI layer is in charge of displaying all the information from the game system layer below to the user. We believe that having a layer for each specific role would allow for greater decoupling and thus easier testing which will be beneficial in the development of our software.