Core architecture premise

A desktop game which may connect to the internet if the user wishes to advance their profile or compare their score to other players all over the world.

Sub-architectures

Client-server Architecture

Centralized Server

The client-server architecture is one of the core architecture design points our group chose to include for our game. The server on which the application uses acts as a centralized point of communication between it and every single client. This means that for all client requests, each request will be routed to the same singular server which will be responsible for handling storage, validation, and processing. A diagram showing how a centralized server interacts with clients can be seen on diagram 1.

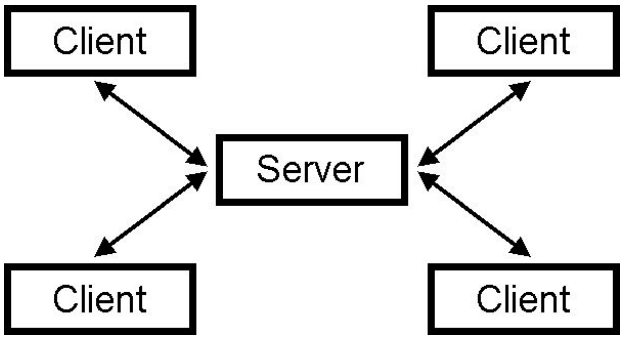

Diagram 1

Client-Server communication

The clients in the application use the provided services offered by the server, and usually are the ones that invoke communication with the server. For example, if a user wishes to compare scores with other users, he/she will look up their scores on the scoreboard that was provided to them by the server. The client in this instance requests a scoreboard of the users from the server which it provides, and now the user can view his/her position on the leaderboards.

Besides providing leaderboards, the client also communicates with the server for three other critical features, login, account creation, and score storage. For each of these services the client will be the one that initiates communication with the server and sends the necessary data. The server in this must provide validation, storage, and authentication of the client's requests
Other than communicating with the server the client is also responsible for actually running the snake game on the users computer and allowing the user to interact with said game.

Advantages/Disadvantages with Server architecture

There are several advantages for using a centralized server to handle the backend of our game. One advantage of having a centralized server is the simplicity and maintainability that comes with it, since all our processing comes from one point it is much easier to maintain and bugfix if anything should go wrong. Having multiple servers can increase the complexity of your system since this introduces a new point of failure (inter-server communication). However, having just one server does limit the scalability and recoverability of a system. As opposed to being able to simply add more servers to our system when our amount of players increases, we are forced to stick to one server for all our processing needs. This means our system is not horizontally scalable but only vertically scalable. We do not expect our server to be under any heavy load anytime soon since our playerbase is relatively small and the requests our server handles are not particularly demanding.
Another advantage of having a centralized server for our game, is that it allows us to perform validation and verification on the server-side as opposed to handling it client-side increasing our overall integrity. This creates a more secure environment since we can handle login and account creation details on the server as opposed to the client having to deal with storing validation for accounts. However this does not completely block us from all security-related obstacles as the server has to deal with making sure the communication it receives from the client is correct and not malicious (SQL Injections).

MVC Architecture

The application uses the MVC (Model View Controller) architectural pattern, meaning it has a division between the Graphics (View), User inputs (Controller) and Back-end for game logic and server requests (Model). This is a software-architectural design pattern, meaning that it will not result in any noticeable difference in user experience. This design pattern seemed the most suitable to keep the software structured and consistent over the entire project.

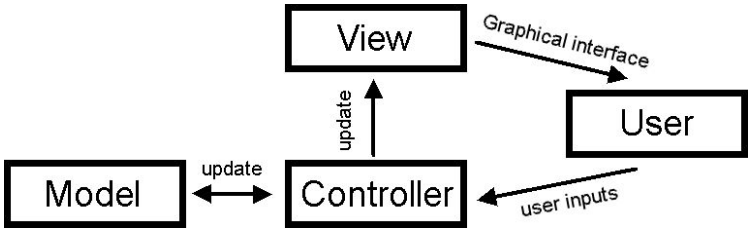

Diagram 2