**(a) Cheating in online games is the action of pretending to comply with the rules of the game, while secretly subverting them to gain an unfair advantage over an opponent. Describe two ways of cheating. Please include details of how each cheating works and a mechanism to prevent it.**

Using DLL injections, cheaters can “hook” onto processes to gain an advantage over other users. Hooking is a scheme where a hack is notified when a specific function is called. Using this mechanism, a Speed Hack can be used by cheaters. This is due to many games using GetTickCount or the High Performance Counter to determine lengths of time, meaning that cheaters can get into the internals of how the functions work to increase how often simulations are updated, and how many packets they can send.  
  
However, Speed Hacks can be caught by using an internal timer server-side, and a timer client-side. If these two timers extend past a certain point and become too far apart, it’s very easy to determine that the player is cheating.  
  
Wall Hacks can also be achieved by “hooking” with DLL injections into openGL drawing methods. For example, Half Life uses the glBegin method to draw it’s quads and triangles. If you can write a program to “hook” into that method, you can tell the game to simply draw the triangles over quads, as the game uses triangles to draw player meshes. This will cause their models to become visible over all walls, allowing you to see through walls.

This type of hack is a lot more difficult to detect, and usually relies on Player reports to catch cheaters themselves. However, one method could easily be to check and see if the cheater should be able to see the target they are looking at. If they seem to stare at the other opponent as they walk behind a wall, it may be an indicator of cheating. You could then add them to a list to check on later to see if reports appear for the cheater.

**(b) Interest management is important for good network performance in massively multiplayer games. Interest management cuts down the bandwidth usage by filtering irrelevant updates. What is a potentially visible set, and how does this approach differ from static zones? How do these interest management approaches benefit the game?**

Potentially Visible Sets are used to accelerate the rendering of 3D environments and is a form of occlusion culling. PVS as a result, is a set of regions that are potentially visible.

In a static zone approach, only objects within a loaded static zone are shown. However, in PVS approach neighbouring regions that are deemed potentially visible will contain relevant objects, instead of having to only load those relevant objects once the zone requires them. The region sizes in PVS are typically much smaller than separate static zones. A static zone might be a town of several buildings, while a PVS region would be an individual room inside of a building.

The goal of interest management is to reduce the cost of data communication distributed within a game. Using interest management improves the scalability of a game by limiting the amount of information given to all players based on relevance.