Lab 9

Lukas kmitas

Q1 - Use an example (e.g., shooting round corners/fire-proof problem) to illustrate how latency is related to consistency? Use diagrams if appropriate.

Q2 - There are two categories of techniques for consistency management: optimistic and conservative. Which category do the following techniques belong to: dumb client and client-side prediction? Describe the dumb client and client-side prediction.

Q3 - 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.

Q1:

**Example**: Imagine you are playing a multiplayer shooter game where players can shoot at each other and take cover behind walls. In this scenario, one player (Player A) tries to shoot another player (Player B) who just moved behind a corner for cover.

**Latency:** Latency, in gaming terms, refers to the delay between the action taken by a player and when that action is registered by the game server and reflected back to all other players. This is often measured in milliseconds.

**The Problem Illustrated:**

1. **Player A's Perspective:**  
   Player A sees Player B running towards the corner and shoots before Player B disappears behind it, due to latency, the shot is registered on Player A's local machine but takes some time (say 100ms) to reach the server.
2. **Player B's Perspective:** Player B successfully reaches the corner and goes behind it, from their perspective, before seeing Player A's gunfire. On Player B's screen, they are safely behind the corner when the shots were fired.
3. **Server's Role:** The server receives Player A's shot data 100ms later. Depending on the server's reconciliation mechanism, it might register the shots as having hit Player B, despite Player B having moved behind cover on their own screen.

**Diagram**

|<-- Latency (100 ms) -->|

Player A -------- shoots ---------> Server ---------> Player B

(Shoots at B) (Processes shot) (Sees cover)

Player B ------ moves to cover --> Server

(Runs to cover) (Registers move)

**Inconsistency Due to Latency:**

**Inconsistency**: From Player A's perspective, the shots should hit because they were fired before Player B reached cover. However, from Player B's perspective, they were already safe when the shots were fired. This inconsistency in game state, caused by latency, can lead to a frustrating experience for players, as the outcome (hit or miss) might seem unfair or arbitrary.

**Impact on Game Play**: This latency can affect how players perceive the game's fairness and responsiveness. High latency can lead to more of these discrepancies, while lower latency helps in achieving a more consistent and synchronized gameplay experience.

**Conclusion**:

Latency directly impacts consistency in multiplayer games by introducing discrepancies between what different players see and what actually happens according to the server's game state. Effective handling of these latency issues, through mechanisms like lag compensation, is crucial for maintaining a fair and enjoyable gaming environment.

Q2:

**Categories of Consistency Management Techniques:**

**Optimistic Techniques**: These techniques assume that most actions will succeed without conflicts and handle inconsistencies only when they occur. They often provide a very responsive experience to the user by immediately reflecting the player's actions locally, then correcting those actions later, if necessary, when the server updates the game state.

**Conservative Techniques**: These techniques take precautions to prevent inconsistencies before they occur. They often involve waiting for confirmation from a central authority (like a game server) before actions are finalized on the client side. This can result in a less responsive but more consistent experience, as actions are only shown to the player after they have been validated.

**Classification of the Techniques:**

**Dumb Client:**

**Category**: Conservative

**Description**: A dumb client architecture in gaming refers to a setup where the client (the player's gaming system) does almost no processing related to the game state or logic. Instead, it merely sends inputs (like keystrokes or mouse actions) to the server, and the server processes these inputs, updates the game state, and then sends back the results to be displayed. This approach minimizes the risk of discrepancies between the game state seen by individual players because all critical computations and decisions are made centrally by the server.

**Client-Side Prediction:**

**Category**: Optimistic

**Description**: Client-side prediction is a technique used to improve the responsiveness of online games, particularly in environments with noticeable latency.

This method predicts the outcomes of actions initiated by a player before receiving the server's response. For instance, when a player moves or shoots, the game immediately shows this action as having occurred, without waiting for the server to confirm it. This can make the game feel more responsive. However, discrepancies can arise if the server's eventual response differs from the prediction, requiring the game to correct the state shown to the player.

**Summary**:

**Dumb Client** uses a conservative approach, relying on the server for all authoritative decisions to maintain consistency. This method reduces the likelihood of discrepancies but can feel less responsive.

**Client-Side Prediction** uses an optimistic approach, enhancing responsiveness at the risk of occasional inconsistencies that must be corrected once the server's authoritative updates are received.

Q3:  
  
**Aimbotting**

**How it Works:**

Aimbotting is a form of cheating where a player uses a program or script that automatically aims at targets, usually in shooter games. This tool allows cheaters to shoot with inhuman accuracy and reaction times, giving them an unfair advantage over other players. The aimbot program analyses the game data to detect other players' positions and instantly aligns the cheater's aim with these positions, often even accounting for game physics like bullet drop or travel time.

**Prevention Mechanism:**

Client Integrity Checks: Game developers can implement systems that regularly check the integrity of game files and memory states of the client machine. If the checks find unauthorized modifications or programs interacting with the game, it can flag or ban the offending accounts.

Heuristic Analysis and Player Reporting: Developers can also use heuristic behaviour detection algorithms to identify patterns typical of aimbots, such as unrealistically quick aiming or perfect accuracy over multiple games. Combined with a robust player reporting system, this can help identify and penalize cheaters effectively.

**Wallhacking**

**How it Works**:

Wallhacking refers to the use of software that enables a player to see through solid objects, such as walls or obstacles, which would normally obstruct a player's view. This cheat gives users an unfair advantage by allowing them to anticipate an opponent's actions without being seen. Wallhacks manipulate the game's rendering processes to make walls transparent or to highlight player models through walls.

**Prevention Mechanism:**

Server-Side Validation: One effective way to combat wallhacks is by limiting the information sent from the game server to the client. The server should only send information about other players' locations when they are realistically perceivable by a player, based on their current position and line of sight.

Techniques: Another method is to use software obfuscation techniques to make it harder for cheaters to understand or alter how the game handles and renders data. This makes developing a wallhack more difficult and time-consuming.