1. Describe the difference between a task and a coroutine in Unity.

A task in Unity is a unit of work that runs on a separate thread from the main Unity thread. It allows for concurrent and asynchronous execution, improving performance in CPU heavy tasks. A coroutine, on the other hand, is a function that allows you to pause its execution and resume it on the next frame or after a specified time. It runs on the main Unity thread, thus it doesn't benefit from multi-threading but allows for simpler code structure and better integration with other API's.

2. What do you feel are common mistakes developers make when building games in Unity?

Common mistakes include poor project organization, not optimizing for performance, overusing GameObject.Find() calls, and neglecting the importance of proper asset management.

3. Why do you think developers make those mistakes?

These mistakes often occur due to a lack of experience, time constraints, or insufficient knowledge of best practices and Unity tools available.

4. Design an Object model for implementing a card game. Provide code for common operations.

This is a simplified object model for a card game:

```
public enum Suit { Hearts, Diamonds, Clubs, Spades }
public enum Rank { Two, Three, Four, Five, Six, Seven, Eight, Nine, Ten,
Jack, Queen, King, Ace }

public class Card
{
    public Suit Suit { get; private set; }
    public Rank Rank { get; private set; }

    public Card(Suit suit, Rank rank)
    {
        Suit = suit;
        Rank = rank;
    }
}

public class Deck
{
    private List<Card> cards;

    public Deck()
    {
        cards = new List<Card>();
```

```
        foreach (Suit suit in Enum.GetValues(typeof(Suit)))
        {
            foreach (Rank rank in Enum.GetValues(typeof(Rank)))
            {
                cards.Add(new Card(suit, rank));
            }
        }
    }

    public void Shuffle()
    {
        Random rng = new Random();
        int n = cards.Count;
        while (n > 1)
        {
            n--;
            int k = rng.Next(n + 1);
            Card temp = cards[k];
            cards[k] = cards[n];
            cards[n] = temp;
        }
    }

    public Card Draw()
    {
        if (cards.Count == 0) return null;

        Card topCard = cards[0];
        cards.RemoveAt(0);
        return topCard;
    }
}
```

5. How would you go about building a computer player for a two-player card game?

To build a computer player for a two-player card game, I'd follow these steps:

a) Create a simple AI that plays by the game rules, making random actions or card plays without any specific strategy.

b) Improve the AI with a goal-based or utility-based approach. With utility-based AI, each potential move gets a utility score that shows how good the move is. The AI evaluates the game state and calculates utility values for all possible actions, choosing the move with the highest score.

I would also keep these factors in mind when evaluating moves:
- Depending on the game, the AI should either focus on attacking the opponent or protecting its own assets.
- Look at the opponent's moves to predict their strategy, which helps make better decisions.
- Check the overall game state and adjust the strategy accordingly, focusing on winning when ahead and catching up when behind.
- Think about other aspects of the specific card game like card probabilities, bluffing, or time constraints.

c) Test and update the AI based on gameplay results. Fine-tune utility functions and factors, or add new strategies to boost the AI's performance in the card game.

d) For a more advanced and adaptive AI, I would also consider using machine learning techniques like reinforcement learning. The AI agent learns the best strategies by playing the game multiple times, enhancing its decision-making abilities through experience.