**STUDENT NAME: Saidi Reddy Vangala**

**TOPIC: ARCHITECTURE USED IN GAMES**

There are many architectures used in developing games, but here are some of the most common ones:

**Client-Server Architecture:** This is the most common architecture used in online games. The game client (the player's computer or device) communicates with a central server, which manages the game world and sends updates to all clients. This architecture is good for games with a large number of players and a persistent game world.

Peer-to-Peer Architecture: In this architecture, each player's computer communicates directly with the other players' computers. This can be good for small games with a low number of players, but it can be difficult to manage and can lead to cheating.

**Hybrid Architecture:** This is a combination of client-server and peer-to-peer architectures. The game uses a central server to manage the game world, but some game data is sent directly between clients. This can be a good compromise between the two architectures.

**Cloud Gaming Architecture:** In this architecture, the game runs on a remote server and the player's device streams video and audio. This can be good for games that require high-end hardware and can reduce cheating.

**Microservices Architecture:** In this architecture, the game is divided into small, independent services that can be developed and deployed separately. This can make it easier to maintain and scale the game, but it can be complex to set up.

Ultimately, the choice of architecture depends on the specific requirements of the game and the resources available to the developers.

**Introduction**

Multi-part series in a game like tic tac Toe is much related to two players on a 3x3 board. This game uses the suitability of using the respective marks of X and O in vertical as well as horizontal lines. There are also diagonal lines that are involved within this game (Klosowski, 2019). The MVC architecture is also known as the model view controller is identified as a particular pattern that necessarily uses software design through its implementation on user interfaces using control logic and data (Al-Awadai, 2019). This game elaborates on the use of MVC architecture. A generic understanding here for this project under eyes the use of the MVC architecture for this tic tac toe game. Most important, leave and define the precise moves and the idea of continuing with the permutations. This is all about the terminal configurations that are determined. This game introduces a perfect modification and the major contributions made using the algorithms of JAVA. Some of the basic aspects that are introduced in this project are the use of random X and O marks.

**Project scope**

The major scope of this project is to study and understand a game like a Tic Tac Toe & The MVC architecture used to the implementation of this game.

(importance of mvc in games development

The Model-View-Controller (MVC) architectural pattern is important in game development because it provides a clear separation of concerns and makes it easier to manage complex systems. Here are some of the specific benefits of using the MVC pattern in game development:

**Separation of Concerns:** By separating the game logic (the model), the user interface (the view), and the input/output handling (the controller), the code is easier to understand, modify, and maintain. Each component of the MVC pattern can be developed and tested independently, and changes to one component are less likely to affect the other components.

**Code Reusability:** Because the MVC pattern promotes modularity, code can be easily reused across different games or different parts of the same game. For example, the same model could be used in different views to display the game state in different ways, or the same controller could be used for multiple games.

**Scalability:** As games become more complex, it becomes more important to manage the different components of the game effectively. The MVC pattern makes it easier to scale the code and to add new features without introducing bugs or making the code harder to maintain.

**Debugging and Testing:** By separating the different components of the game, it becomes easier to debug and test the code. It is easier to isolate bugs and to test specific components without affecting the rest of the game.

Overall, the MVC pattern is a powerful tool for game development. It allows developers to manage complex systems effectively and to build scalable, maintainable codebases that can be easily modified and extended over time)

**Project description**

The game Tic Tac Toe is identified here as multi-part series along with the implementation of the self-learning nature of the game. Most importantly this game relates to different Technology and their comparisons. The architecture that is introduced here is the MVC architecture for the running of this game. Most important when defining the program designs it is also about the interface where the game is involved. There are controller classes along with views to be determined for defining the graphical user interface of the game. With the use of Java and the embassy design for making the game, it is all about creating winning conditions and losing conditions for 2 users.

(Tic Tac Toe can be developed using the Model-View-Controller (MVC) architectural pattern. Here's an example of how the MVC pattern can be used in Tic Tac Toe:

Model: The model represents the state of the game and the game logic. It keeps track of the positions of X's and O's on the board, checks for a win or a tie, and notifies the controller when a move has been made.

View: The view represents the user interface. It displays the game board and allows the player to make moves. It receives input from the player and passes it to the controller.

Controller: The controller acts as the intermediary between the model and the view. It receives input from the view, updates the model, and updates the view with the new state of the game. It also handles the logic for the computer AI, if present.

Here's an example of how the interactions might work in the MVC pattern:

The player makes a move by clicking on a square in the view.

The view sends a message to the controller indicating which square was clicked.

The controller receives the message and updates the model with the new move.

The model checks for a win or a tie and notifies the controller of the result.

The controller updates the view with the new state of the game, displaying any changes in the positions of X's and O's and displaying the winner or a tie message.

By using the MVC pattern, the code for the game can be more modular and easier to maintain. The view can be easily updated without affecting the underlying game logic, and the model can be modified without affecting the user interface.)

**Architectural Styles and Patterns**

The basic architectural style that is used here introduces the controller class along with the view class and the model class. These three are necessarily important for aggregating the entire game of tic tac Toe. There are logics and results for winning which come through the graphical user interface. Most importantly the view class becomes extremely necessary for the displaying of the entire game (Al-Awadai, 2019). The architecture of this game is prepared through Java and and also introduces adaptor classes where the controller classes are underlined and instigated for appropriate action. Several basic criteria are also introduced here for this particular game project. The architecture of MVC is hence defined for the interface and the update state. The game of tic tac Toe uses the controller classes along with the model and view for defining the entire winning logic and also for interacting with the graphical user interface (Klosowski, 2019). This is the basic architecture and its styles that are defined by the Java implementation of the MVC architecture approach.

**References**

Al-Awadai, Z., Brüggemann-Klein, A., Grubmüller, C., & Ulrich, P. (2019, July). Graphical user interfaces in the X stack. In Balisage: The Markup Conference.

Klosowski, K. (2019). Chess Engine Using Deep Reinforcement Learning.