Architecture of Quest of the Round Table

Team 20

The game was developed with MVC in mind. The model would be updated by the controller “MultiplayerGame.cs” and “Game.cs” respectively depending on the mode you were in. These game controller .cs files stated above instantiate the models on initialization and update them depending on the actions done on the view. We also used the factory design pattern in the need of serialization for multiplayer, the factory file is called “CardFactory.cs” and was created with the intent to cards depending on other client’s actions in multiplayer. All card objects use inheritance with the super class Card in “Card.cs”, “AdventureCard.cs” and “StoryCard.cs”. We also implemented the Singleton pattern on “MultiplayerGame.cs” and “Game.cs”.

The core of general game logic lies in “MultiplayerGame.cs” and “Game.cs”. These files keep track of who’s turn it is in the general game sequence (not tournaments and quest) and are singleton objects that get instantiated on scene load. All sub game mode logic lie in these respective files “MultiEventBehvaiour.cs”, ”MultiQuestBehaviour.cs”, “MultiTournamentBehaviour.cs”,” EventBehvaiour.cs”,” QuestBehaviour.cs”,and “TournementBehaviour.cs” and use the strategy pattern and get their respective “endturn” methods invoked by the game controllers depending on which story card is currently active on the field. The AI also implements the strategy pattern and it’s strategies can be found in “AbstractAI.cs”, “AIBehaviour.cs” , “ParticipateInQuest.cs”, “ParticipateInTournament.cs”,”SponsorQuest.cs”, and “TestBid.cs”.

All networking is handled by Photon in Unity in “PhotonNetworkManager.cs” which links all the clients trying to join the game together. Messages get passed via “PunRPC” calls which invoke methods on each different unity client with serialized primitive values. The RPC calls can be found in the “MultiplayerGame.cs” and “PhotonEventHandler.cs”. When these functions are called they invoke the same function call on all other clients. This architecture essentially brings multiplayer to the game by moving all player clients through different states depending on which “PunRPC” calls are invoked.

Server-Client architecture of “Quest of the Round Table”

Team 20

The server client architecture of this game is done via a unity solution called Photon. Photon is a service that lets us host our game on the Photon Network. All connections to this server the beginning of the game is handled by “PhotonNetworkManager.cs”. This class just connects all clients to the server. Throughout the game the clients will call “Remote Process Calls” that will invoke the same actions on each other clients a move each client’s through the same states as each others.





Use Cases:

Use Case 1 (Host a Game)

1. User creates a game on the server
2. User allows other Players to join
3. User Starts game

Use Case 2 (Join a Game)

1. User finds the available games
2. User Joins the game
3. User wait for host to start game

Use Case 3 (Sponsor a Quest)

1. User declares the decision to sponsor the quest
2. System notifies all players the User have chosen to sponsor
3. User sets up the stages with their hand cards
4. System verifies the stages and start the quest

Use Case 4 (Join and Participate in a Quest)

1. User declares decision to join quest
2. System notifies all players the User have chosen to sponsor
3. System gives User a card at the start of a stage
4. User Sets up their allies and weapons
5. User discards cards if too much cards
6. System determine if User passed the stage and hand cards not over and proceed to next stage, go to 3
7. System determines that all the stages have ended and award the remaining players shields

Use Case 5 (Tournament)

1. User declares decision to join the tournament
2. User plays cards to their play area
3. System determine who won the tournament
4. System awards shields

Use Case 6 (Events)

1. User draws an Event card
2. System recognize the event card and execute event
3. User discard cards if hand size over
4. User end turn
5. System verifies hand size