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”.

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 the hot seat game mode to multiplayer 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 eachother.s