Protocol

Describes design

- MTCG Game:
    - In this Project the Game Functionalities are separated into individual departments. For starters the Card Class is the Parent Class of Monster and Spell-card, here I decided to use a Class and not an interface because the inheriting classes are very similar in their parameters and function that a lot of repeatability is avoided. These Classes are all found under the Card Directory.
      And the Parentclass Card is used to form other Structures like CardDeck, CardStack and CardPackage which all have a Card List and functions to fill said lists.
      Next there is the Logic Directory where the actual Game Logic is found in. The Class GameLogic implements all the functions for the game loop and round activities. The PackageTrading Class implements functions on what has to happen when a User buys a package.
      Other than these there is a User class of course that has all attributes that are also in the database, id, name, wins, loses, elo, matches, stack, deck, etc.
      There is also a Directory Enums which has a Class Card_Enums that implements the Enums for Monsters as well as Elementtypes.

- MTCG Test:
    - This Project contains all the necessary tests to ensure functionalities of the Gamelogic as well as Damage calculation, specialties, Deck being updated after battle, Card attributes and effects work as intended.
      Also, Class attribute setting as well as none settable attributes are tested that access rights are set appropriately through the getter and setter.

- MTCG Server:
    - The Server is responsible for the DB connection as well as Player Management for the game and lobbying.
      There is a Directory of Endpoints which are classes used for serializing and deserializing, overall, there are 4 classes: User, Card, Package, Trading.
      Another Directory is Requests which contains all end Request separated in User Card Package, Trading and General Request-Classes.
      The Server also needs a way to get Requests, for this task the RequestHandler Class is responsible. Separated from that the BodyHandler and HeaderHandler read their designated parts, if they are available. HTTP_Response is used to either set Unique Responses for the User with code, description and message, or there are preset Responses as well.
      For The General Server start and loop the class ServerLoop has all the functionalities.
      Lastly there is IDFactory which is used to create new ids for cards when they are created and added to the database.

Describes lessons learned

I've already programmed my fair share and projects since the 8 years I'm taught in C#, but nonetheless there is always something to learn. I for example am used to big Classes and used to avoid separating functionalities if the class that would be created is rather small. I've seen though that this job separating approach in really condensed but smaller Classes can work really well and is also pretty in a way. It's not a Design-Pattern that is new to me but just something I noticed during development. Specially in the Projects that we have seen during showcases.

I don't work with Enums often so it was nice to use them too, especially because they made the Card generation more variable and easier to add to in the future. If the Generation should be more randomized there could be an Option to make Enums for Damage as well so values can be more controlled in comparison to random.Range.

For me the work amount was 80% SQL and Server heavy and only 20% actual C#, in the end it felt more like I was programming for a Database Course than C#. Maybe I also feel this way because the C# bit of the Game-Logic was easier for me due to my experience but SQL connection was something newer to me.

I also really saw the help of Unit tests in the project, because I could easily test if my code still worked after I made changes.

Describes unit testing decisions

For Unit testing it was important to me that there are lots of tests for the combinations in battle and that the logic for card comparisons, wins, damage, specialities and effects.

There are Tests for the Specialities between Cards to see that the right winner emerges from the function.
There are Tests for Effect Calculation when there are Only Monster, Spell or Mixed Battles, where the winner is checked. Adding to all these, there are also tests that check that the effects are calculated and applied correctly.

There is an extra User-test-Class for the User tests, which check for the attribute configurations. Same goes for the Card Class.

There are also tests for the Unique feature to see if it behaves as intended, for example the new Element Electro, that I added, has effects with Water and Fire Spell-Cards, which affect the damage calculation.
Overall there are over 30 Unit tests, 35 to be exact.

Describes unique feature

I added a few things for the unique Feature, just for Userstats, I also track wins, loses, and Matches. With this information in the DB, I can calculate winning percents, lose percents, draw percents, as well as their count of course.

For the Game I added a new Element type Electro, which has the Effect for Mixed and only Spell card Battles that if the other Element type is Water, the reaction electrocuted is induced which makes the electro card damage double and the opponents reduce by 75%. If the

Opponents Card is Fire the reaction Explosion happens and opponents Cards Damage is reduced by 40% and Electros by 25%.

I also have a scoreboard for wins which is accessed through /ScoreboardWins.

Contains tracked time

Time: ~50 Hours

Production TimeLine:

- Start of the Semester: GamLogic and Card classes
- December after Testphase: ServerStructure + Rework
- Winter break: SQL code

Contains link to GIT

https://github.com/MariaKnie/Knie_CardProject2023.git