Communication Protocol

Our communication protocol is based on Json messages built by a standard constructor called MessageBuilder which converts a Message into a json string and vice versa. These static methods are called toJson and fromJson.

The messages can also be ActionRequests and Actions.

ActionRequests are messages sent by the server to the client and they require that the client chooses an Action based on the possible ones specified in the request.

Actions are responses to an ActionRequest previously received from the server.

In order to deserialize correctly a message we registered the messages' type in an enumeration with the corresponding message class. Thanks to this, when a new message arrives, we use the method retrieveByMessageClass in order to get the type of the deserialized message and use it to cast the Message to the correct class.

In order to ensure that the connection is still alive, every 5 seconds a CONNECTION_ALIVE Communication Message is sent to each client. Then an OK Communication Message is received from each client if they are still connected.

Example of deserialization:

```
String json = in.nextLine();
Message msg = MessageBuilder.fromJson(json);
switch (MessageType.retrieveByMessageClass(msg)) {
            case CHOSEN_TEAM -> {
                  …
            }
            case START_GAME -> …
           …
}
```

Example of serialization:

```
Message newMessage = new ChosenCloud(3)
String json = MessageBuilder.toJson(newMessage);
out.println(json);
```

Action Messages

ActivatedCharacterCard

This message signifies that the player activated a character card.

```
{
      // index of the character card that was activated.
      characterCardIndex : Integer
}
```

ChosenCloud

This message signifies that the player chose a cloud.

```
{
      // index of the chosen cloud.
      cloudIndex : Integer
}
```

ChooseIsland

This message tells the player to choose an island.

```
{
      // available island indexes.
      availableIslandIndexes : Set<Integer>
}
```

ChosenStudentColor

This message signifies that the player chose a student color.

```
{
      // chosen student color.
      studentColor : StudentColor
}
```

ConcludeCharacterCardEffect

This message signifies that the player concluded the activated character card effect.

```
{
}
```

MovedMotherNature

This message signifies that the player moved mother nature.

```
{
      // number of moves that mother nature made.
      numMoves : Integer
}
```

MovedStudent

This message signifies that the player moved a student.

```
{
      // from where the student was moved
      from : MoveLocation
      // from which index the student was moved.
      fromIndex : Integer
      // to where the student was moved.
      to : MoveLocation
      // to which index the student was moved.
      toIndex : Integer
}
```

PlayedAssistantCard

This message signifies that the player played an assistant card.

```
{
      // assistant card that was played.
      assistantCard : AssistantCard
}
```

SwappedStudents @extends **MovedStudent**

This message signifies that the player swapped two students.

```
{
      // from where the student was moved
      from : MoveLocation
      // from which index the student was moved.
      fromIndex : Integer
      // to where the student was moved.
      to : MoveLocation
```

```
      // to which index the student was moved.
      toIndex : Integer
}
```

ActionRequest Messages

ChooseCloud

This message tells the player to choose a cloud.

```
{
      // available cloud indexes.
      availableCloudIndexes : Set<Integer>
}
```

ChooseIsland

This message tells the player to choose an island.

```
{
      // available island indexes.
      availableIslandIndexes : Set<Integer>
}
```

ChooseStudentColor

This message tells the player to choose a student color.

```
{
      // available student colors.
      availableStudentColors : EnumSet<StudentColor>
}
```

MoveMotherNature

This message tells the player to move mother nature.

```
{
      // maximum movement mother nature can do.
      maxNumMoves : Integer
}
```

MoveStudent

This message tells the player to move a student.

```
{
```

```
      // from location.
      from : MoveLocation
      // from location indexes.
      fromIndexesSet : Set<Integer>
      // to location.
      to : MoveLocation
      // to location indexes.
      toIndexesSet : Set<Integer>
}
```

MultiplePossibleMoves

This message tells the player to move or swap students.

```
{
      //possible moves the player can make.
      possibleMoves : List<Move>
}
```

PlayAssistantCard

This message tells the player to play an assistant card.

```
{
      // playable assistant cards.
      playableAssistantCards : EnumSet<AssistantCard>
}
```

SwapStudents @extends **MoveStudent**

This message tells the player to swap two students.

```
{
      // from location.
      from : MoveLocation
      // from location indexes.
      fromIndexesSet : Set<Integer>
      // to location.
      to : MoveLocation
      // to location indexes.
      toIndexesSet : Set<Integer>
}
```

Client Messages

ChosenGame

This message is a require for a new game with specified settings

```
{
      // Preset is the number of player.
      Preset : GamePreset
      // Mode is the type of the game
      Mode : GameMode
}
```

ChosenTeam

This message is a request for changing team

```
{
      //new team
      tower : Tower
}
```

Login

This message is a request for joining a party

```
{
      //the nickname of who wants to join
      Nickname : String
}
```

StartGame

Message for request the start of a game

```
{
      // Default value of the message
      startGame : boolean
}
```

Server Messages

CommMessage

Send a standard respond to the client

```
{
      //type of the standard respond
      type : CommMsgType
}
```

CurrentGameState

This message is the representation of the game situation

```
{
      //view of the situation
      gameView : GameView
}
```

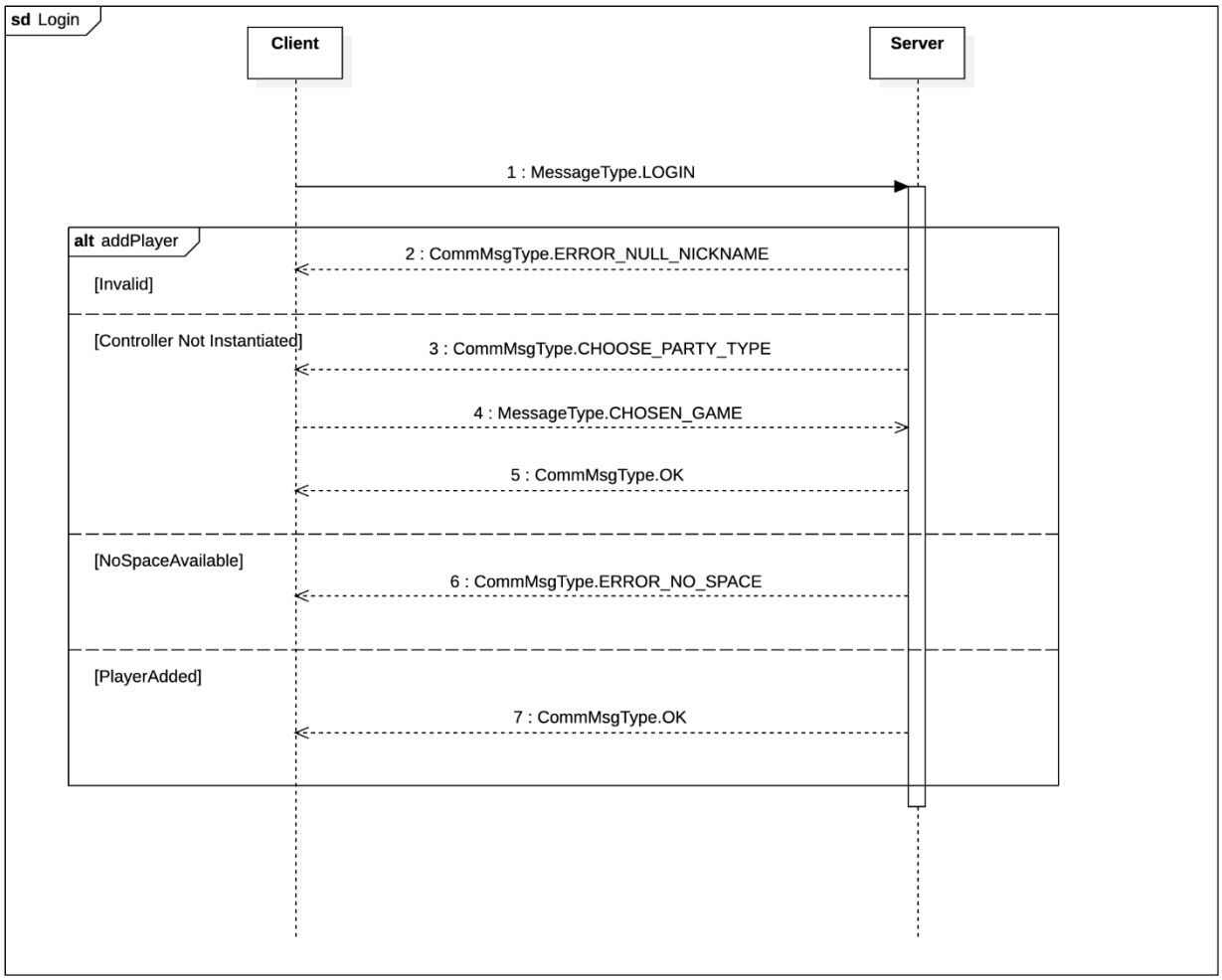
CurrentTeams

Send the composition of the teams

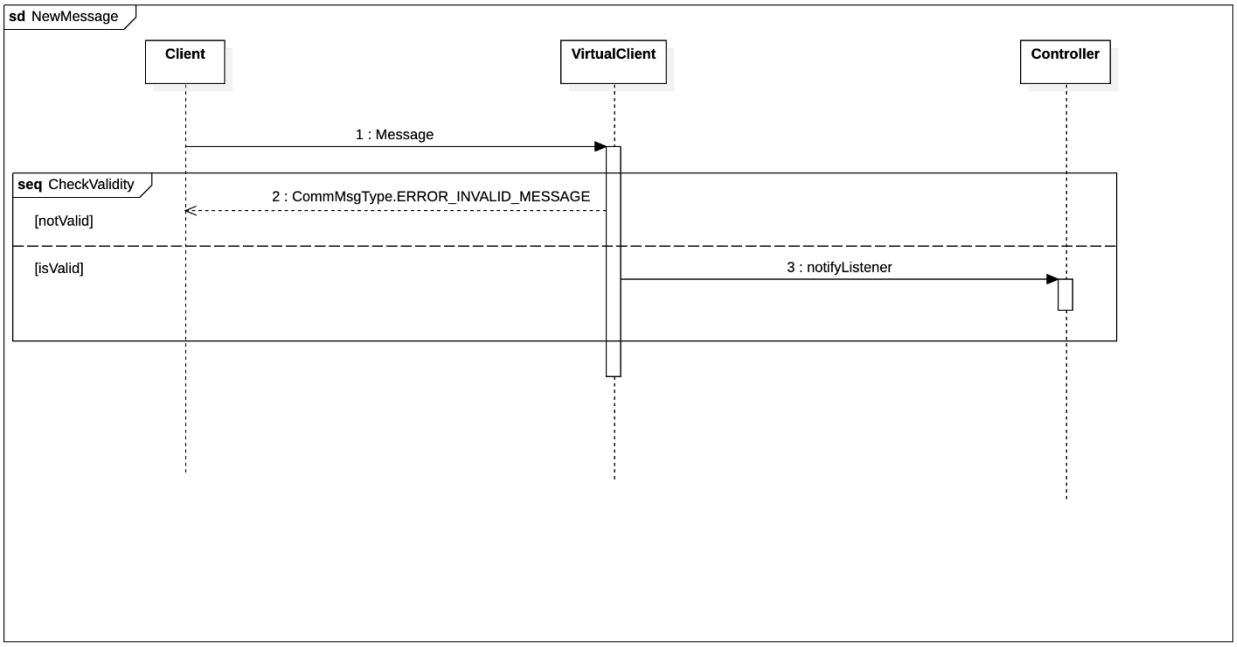
```
{
      //composition of the teams
      teamsView : TeamsView
}
```

Sequence Diagram

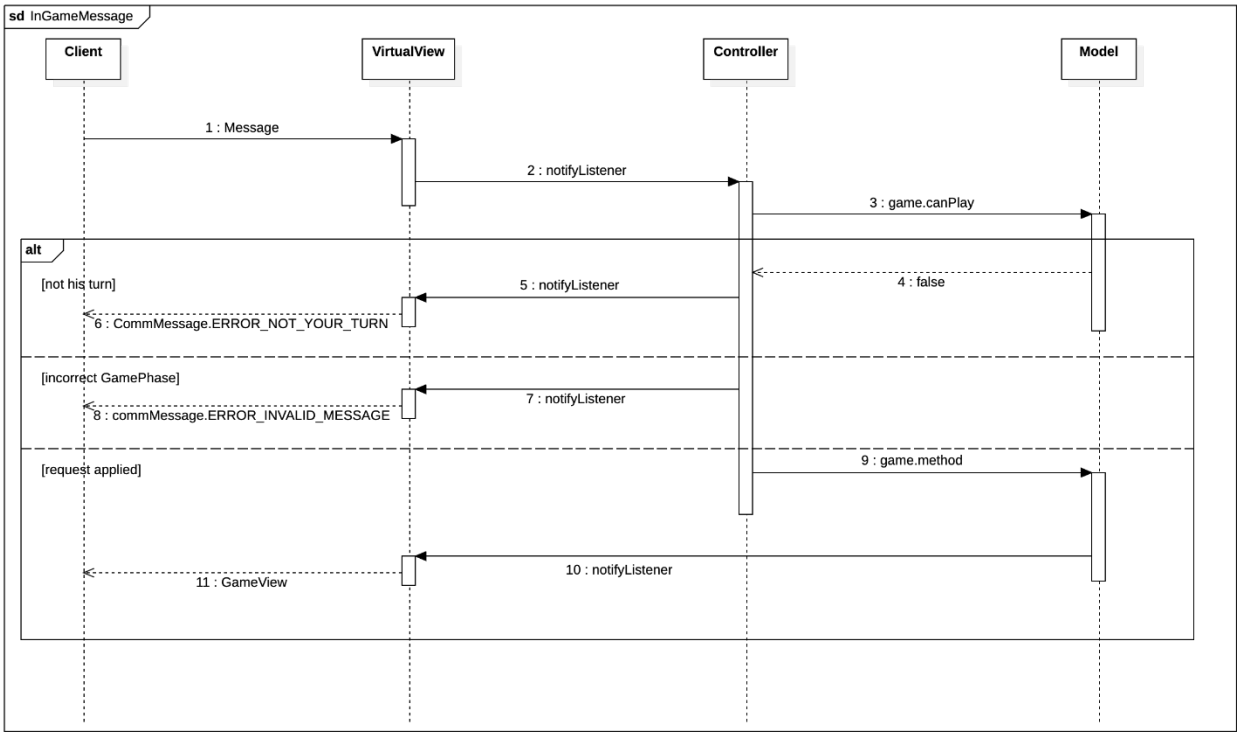
First connection.



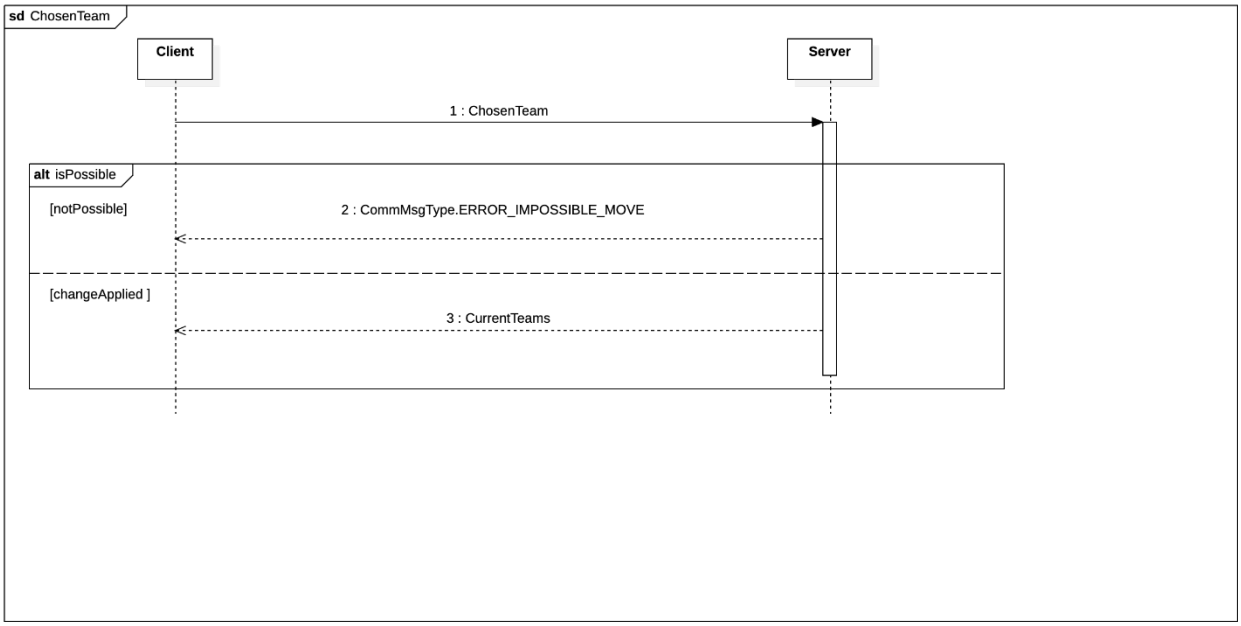
Invalid Message (a message with invalid fields).



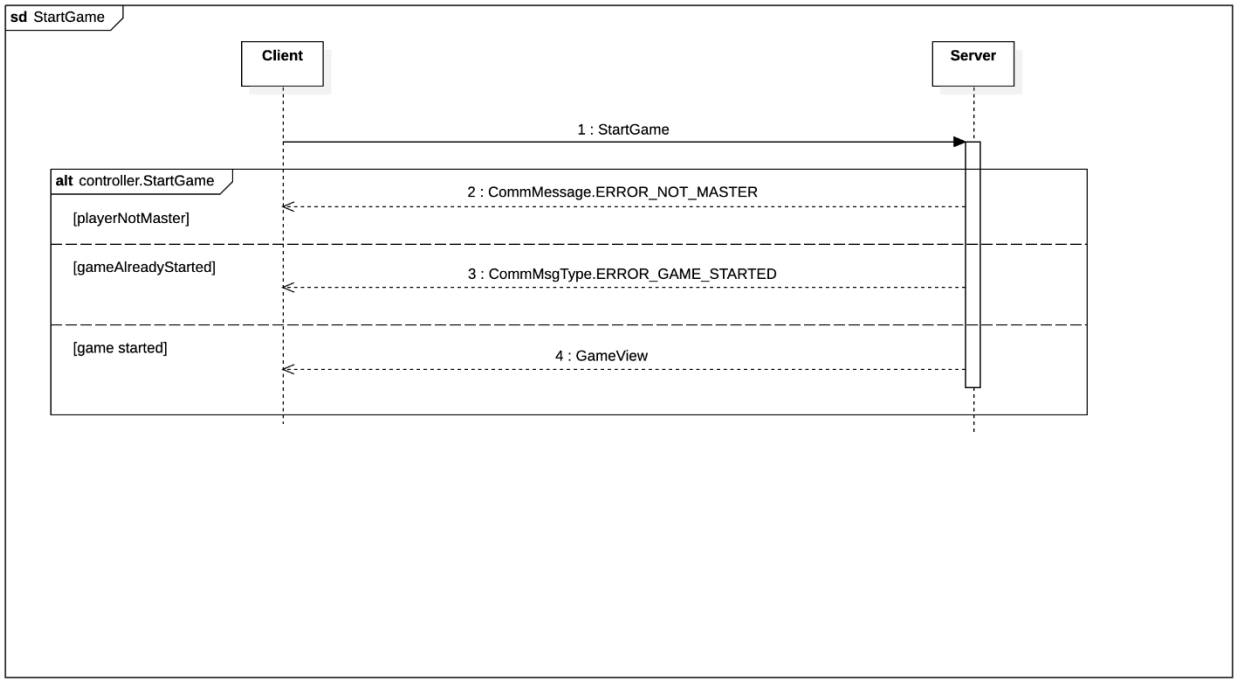
Sequence of calls when a new valid message arrives.



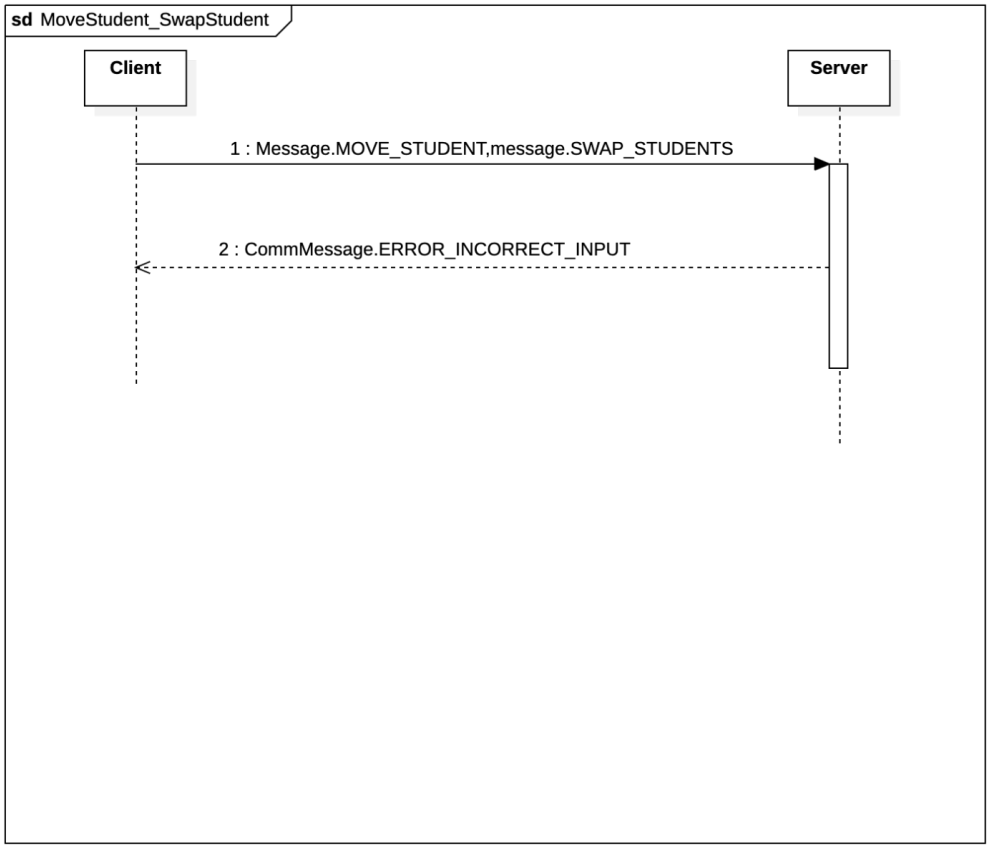
Choose Team Messages.

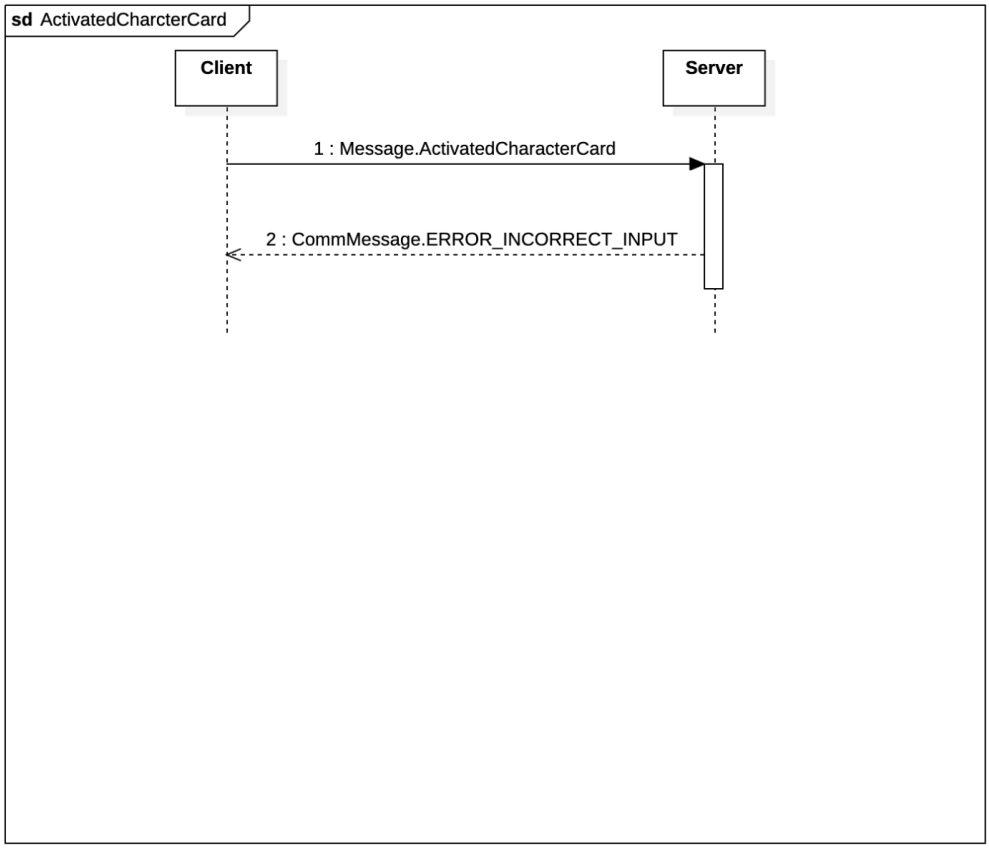


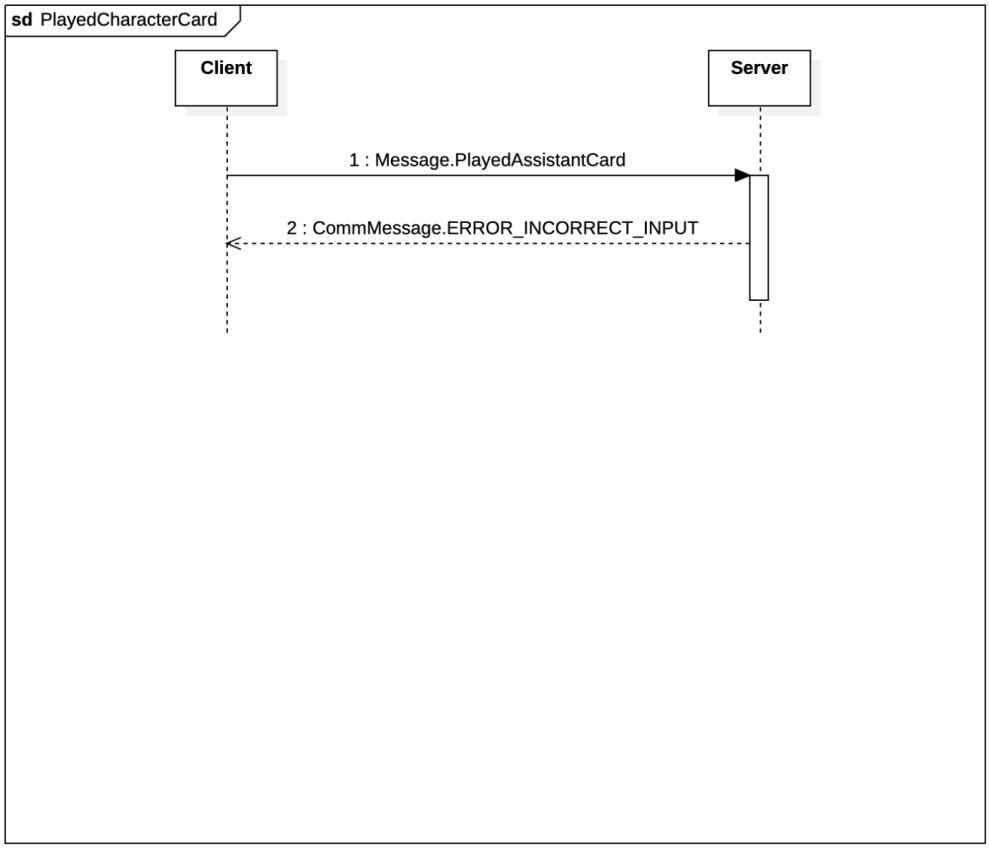
Start Game Message.

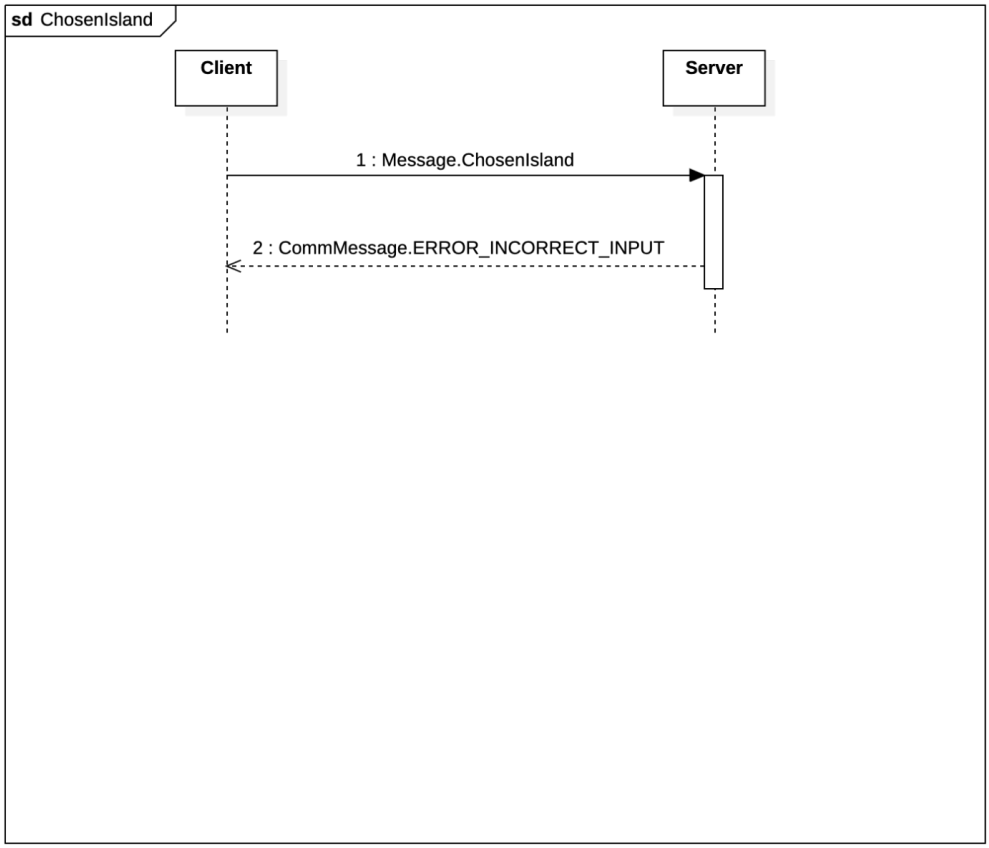


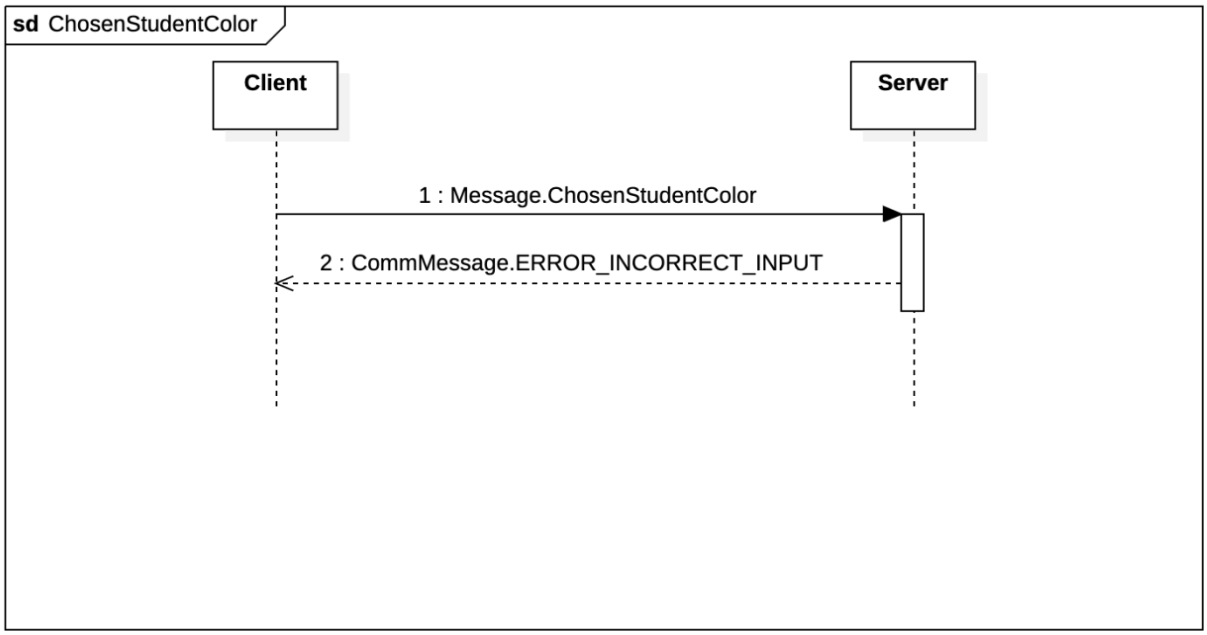
A rapid view on message that can occour when a game is started.

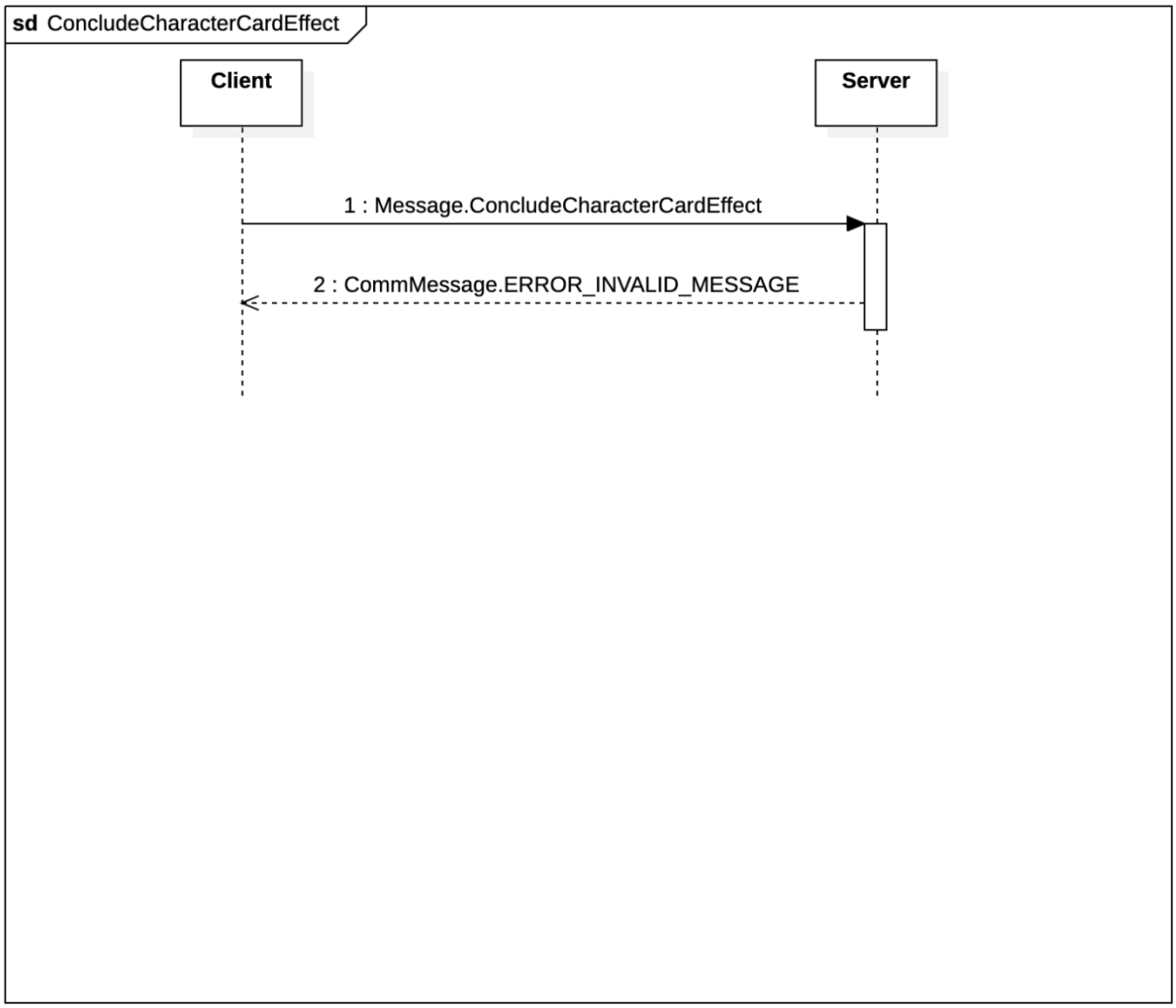


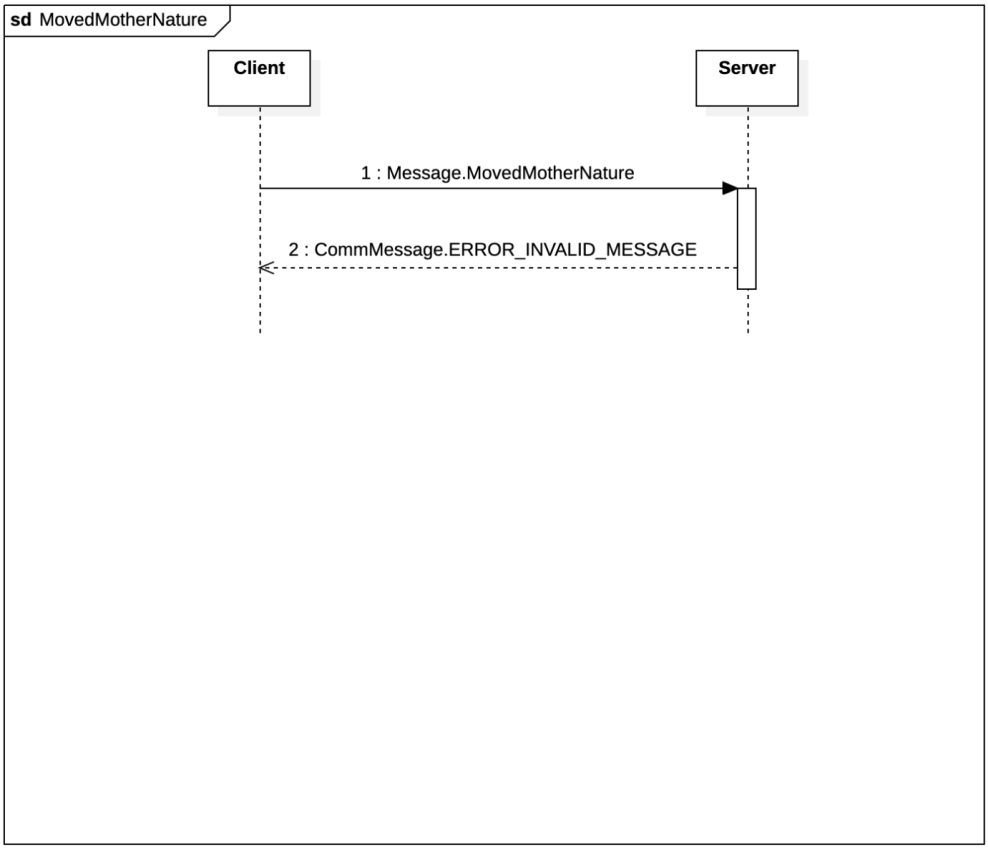


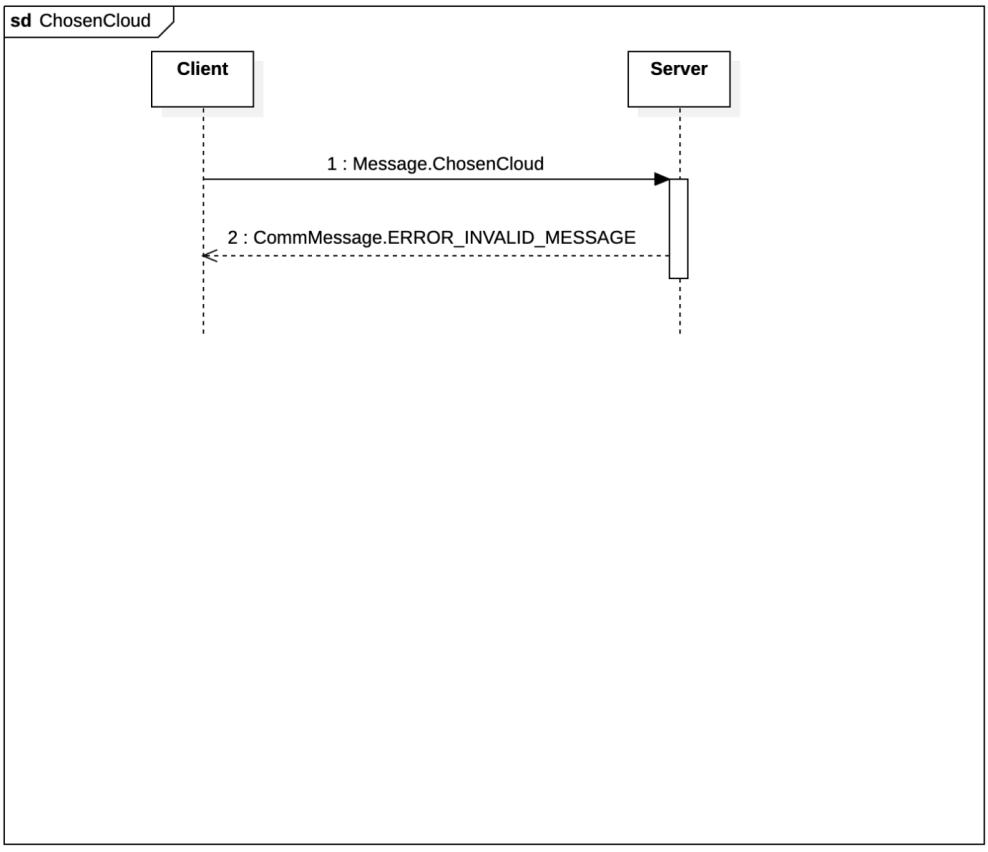

sd PlayedCharacterCard
Client
Server
1 : Message.PlayedAssistantCard
2 : CommMessage.ERROR_INCORRECT_INPUT

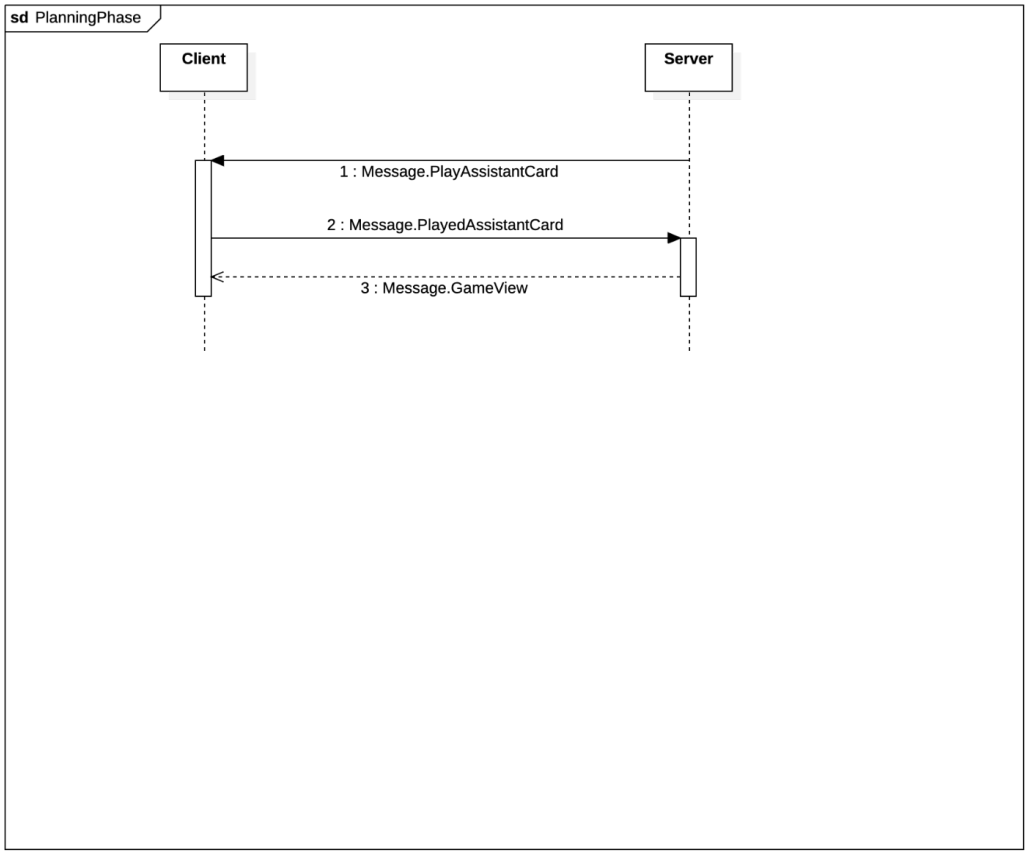

sd ChosenIsland
Client
Server
1 : Message.ChosenIsland
2 : CommMessage.ERROR_INCORRECT_INPUT

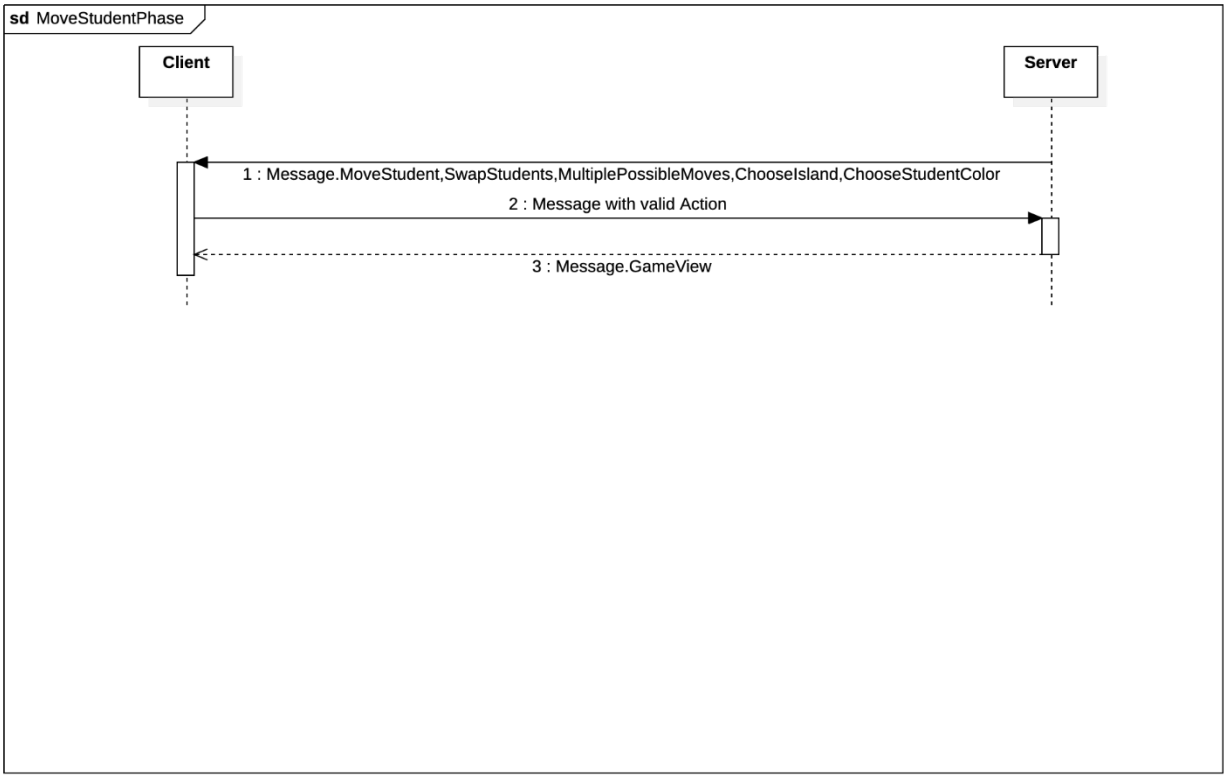

sd ChosenStudentColor
Client
Server
1 : Message.ChosenStudentColor
2 : CommMessage.ERROR_INCORRECT_INPUT

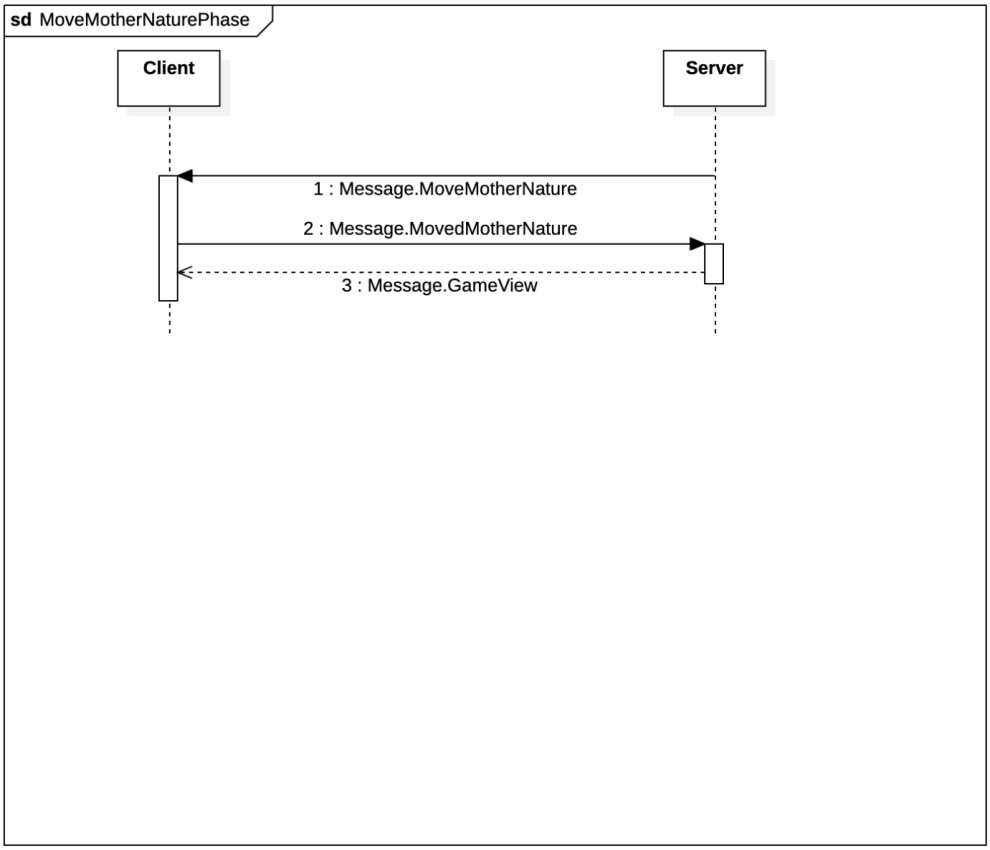

sd ConcludeCharacterCardEffect
Client
Server
1 : Message.ConcludeCharacterCardEffect
2 : CommMessage.ERROR_INVALID_MESSAGE

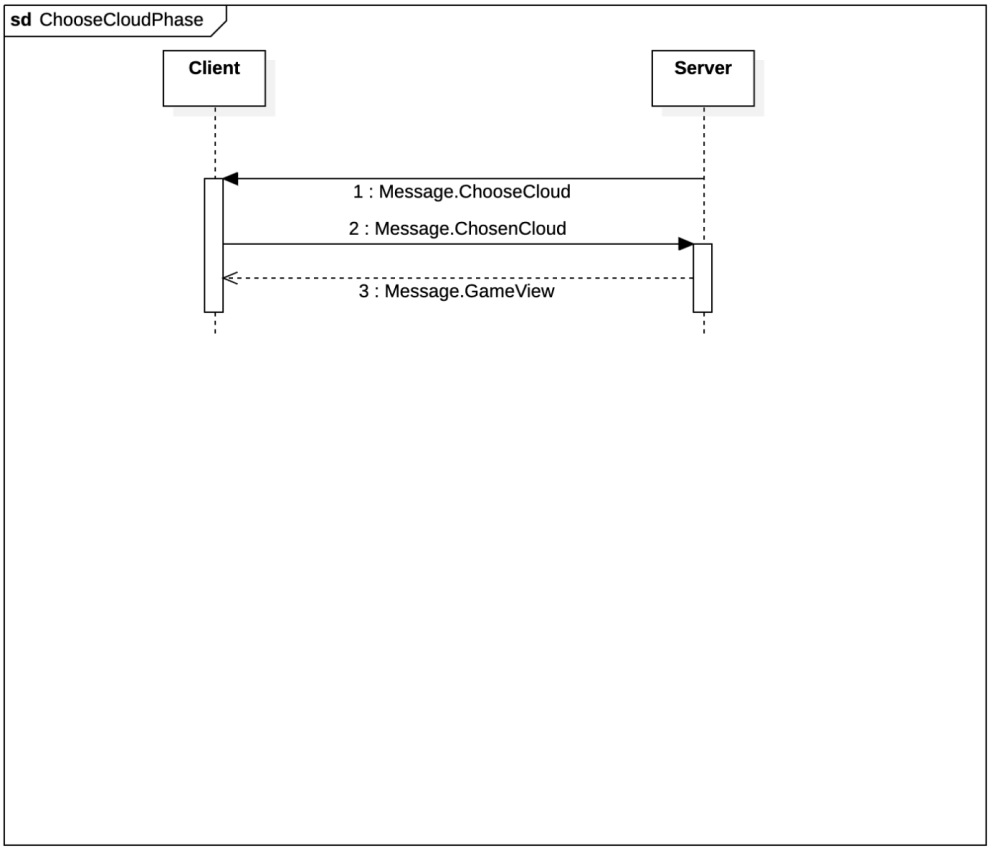

sd MovedMotherNature
Client
Server
1 : Message.MovedMotherNature
2 : CommMessage.ERROR_INVALID_MESSAGE


sd ChosenCloud
Client
Server
1 : Message.ChosenCloud
2 : CommMessage.ERROR_INVALID_MESSAGE


sd PlanningPhase
Client
Server
1 : Message.PlayAssistantCard
2 : Message.PlayedAssistantCard
3 : Message.GameView


sd MoveStudentPhase
Client
Server
1 : Message.MoveStudent,SwapStudents,MultiplePossibleMoves,ChooseIsland,ChooseStudentColor
2 : Message with valid Action
3 : Message.GameView


sd MoveMotherNaturePhase
Client
Server
1 : Message.MoveMotherNature
2 : Message.MovedMotherNature
3 : Message.GameView


sd ChooseCloudPhase
Client
Server
1 : Message.ChooseCloud
2 : Message.ChosenCloud
3 : Message.GameView