Project Report

Table of Contents

Introduction

Among Us is a multiplayer game set on an alien spaceship, typically accommodating a crew of four to ten players. Players are randomly assigned one of two roles: "crewmate" or "imposter." Crewmates are tasked with completing various assignments around the ship, avoiding being killed by impostors, surviving to vote off impostors, and continuing as ghosts if eliminated to aid living crewmates. In contrast, impostors aim to strategically execute murders without arousing suspicion, sow discord among players, eliminate enough crewmates to equal or outnumber remaining impostors, and avoid being voted out during discussions and voting sessions. The game's mechanics include varying the number of impostors based on the player count, allowing crewmates to complete tasks while impostors sabotage and kill, initiating voting sessions upon discovering dead bodies or by agreement, engaging in discussions and accusations during votes to eject potential impostors, and ultimately determining victory based on whether crewmates complete all tasks or vote out impostors, and vice versa for impostors to achieve victory.

About Dataset

The data set has entries 499 Among Us games. There are four high-level fields:

1.Game
2.Game feed
3.Player data
4.Voting data

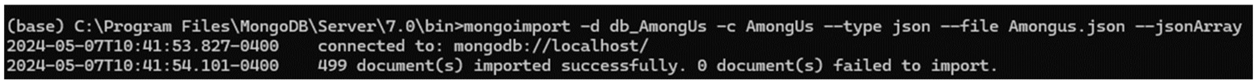
The game is a unique identifier for each game. The other three fields are divided into nested documents with more granular-level data. A game feed is an array of objects where each object is an event in the game. Each game in Among Us has multiple events.

Task 1 – All About Data

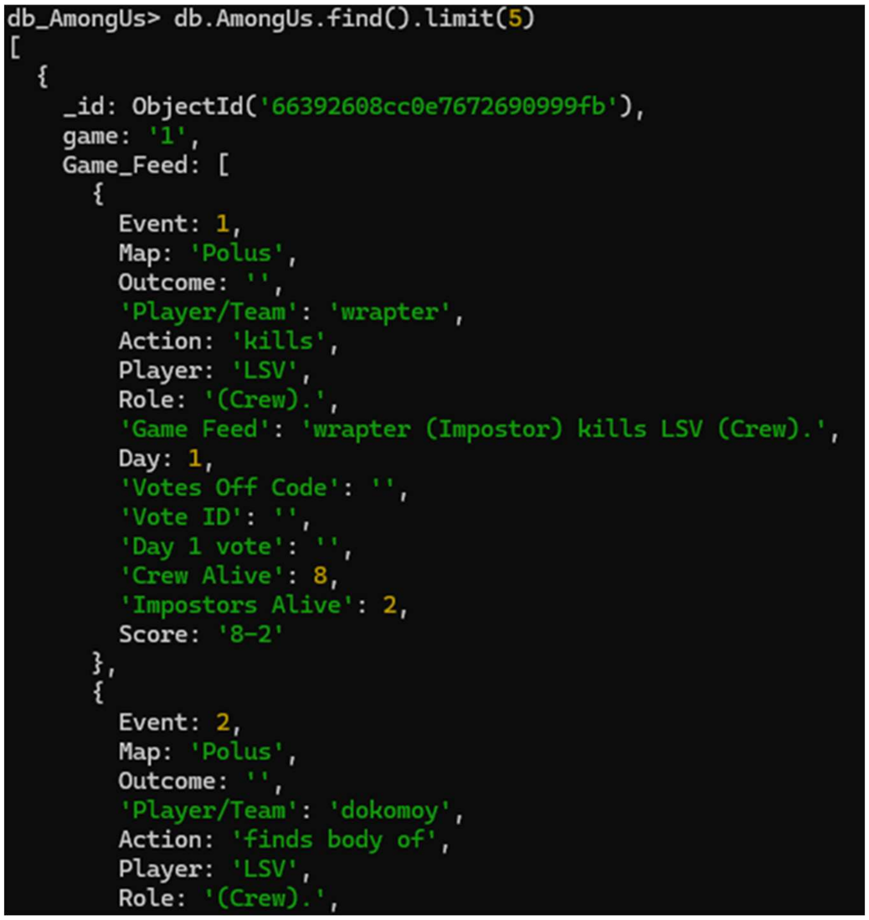
1.1 Read the data and examine the collections

mongoimport -d db_AmongUs -c AmongUs --type json --file Amongus.json --jsonArray

use db_AmongUs
db.AmongUs.find().limit(5)



```
(base) C:\Program Files\MongoDB\Server\7.0\bin>mongoimport -d db_AmongUs -c AmongUs --type json --file Amongus.json --jsonArray
2024-05-07T10:41:53.827-0400    connected to: mongodb://localhost/
2024-05-07T10:41:54.101-0400    499 document(s) imported successfully. 0 document(s) failed to import.
```

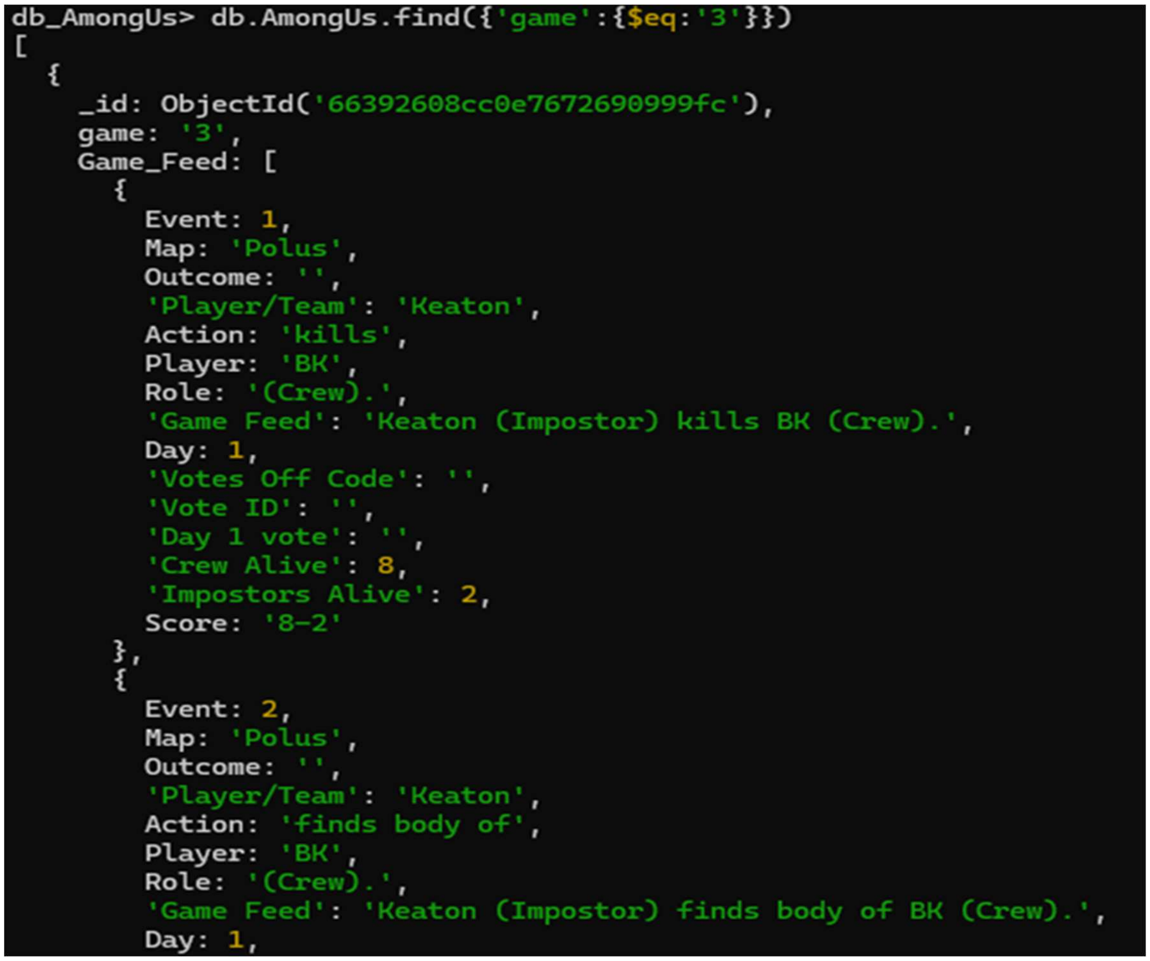


```
db_AmongUs> db.AmongUs.find().limit(5)
[
  {
    _id: ObjectId('66392608cc0e7672690999fb'),
    game: '1',
    Game_Feed: [
      {
        Event: 1,
        Map: 'Polus',
        Outcome: '',
        'Player/Team': 'wrapter',
        Action: 'kills',
        Player: 'LSV',
        Role: '(Crew).',
        'Game Feed': 'wrapter (Impostor) kills LSV (Crew).',
        Day: 1,
        'Votes Off Code': '',
        'Vote ID': '',
        'Day 1 vote': '',
        'Crew Alive': 8,
        'Impostors Alive': 2,
        Score: '8-2'
      },
      {
        Event: 2,
        Map: 'Polus',
        Outcome: '',
        'Player/Team': 'dokomoy',
        Action: 'finds body of',
        Player: 'LSV',
        Role: '(Crew).',
```

For our project, we utilized MongoDB as our database management system, specifically working with a collection named "AmongUs" to store game data. This query retrieves the first five documents from the "AmongUs" collection, providing a glimpse into the type of data we are working with.

1.2. Display data for matches where the "game" field is equal to "3”.

db.AmongUs.find({'game':{$eq:'3'}})



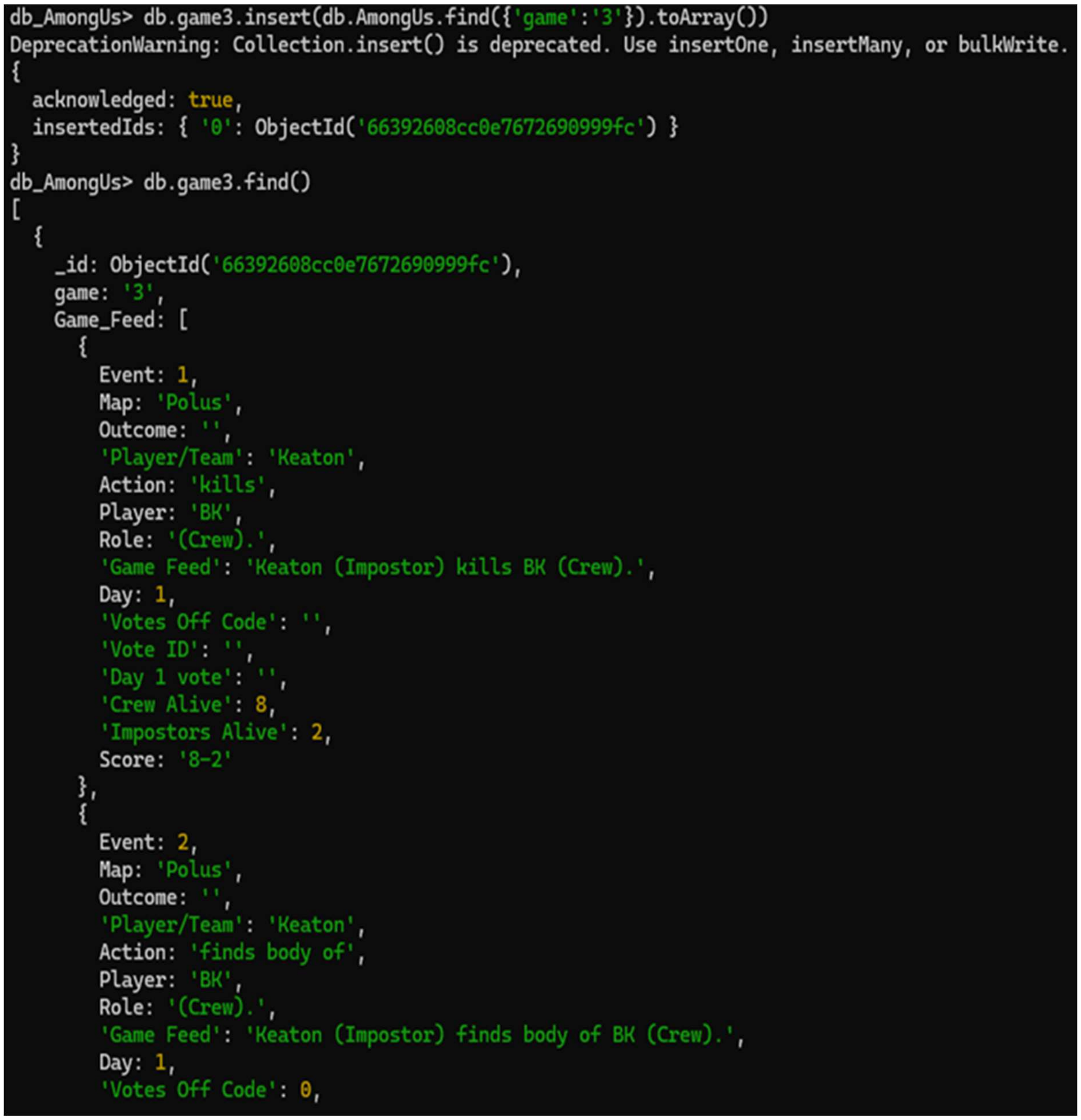
```
db_AmongUs> db.AmongUs.find({'game':{$eq:'3'}})
[
  {
    _id: ObjectId('66392608cc0e7672690999fc'),
    game: '3',
    Game_Feed: [
      {
        Event: 1,
        Map: 'Polus',
        Outcome: '',
        'Player/Team': 'Keaton',
        Action: 'kills',
        Player: 'BK',
        Role: '(Crew).',
        'Game Feed': 'Keaton (Impostor) kills BK (Crew).',
        Day: 1,
        'Votes Off Code': '',
        'Vote ID': '',
        'Day 1 vote': '',
        'Crew Alive': 8,
        'Impostors Alive': 2,
        Score: '8-2'
      },
      {
        Event: 2,
        Map: 'Polus',
        Outcome: '',
        'Player/Team': 'Keaton',
        Action: 'finds body of',
        Player: 'BK',
        Role: '(Crew).',
        'Game Feed': 'Keaton (Impostor) finds body of BK (Crew).',
        Day: 1,
```

The MongoDB query searches the "AmongUs" collection for documents where the value of the "game" field is equal to '3'. It retrieves all documents that match this condition, essentially filtering the data to only include entries related to the game with the identifier '3'

Task 2 – Tasks on Game3 Collection

<u>Create a new collection containing only the records related to game 3</u>

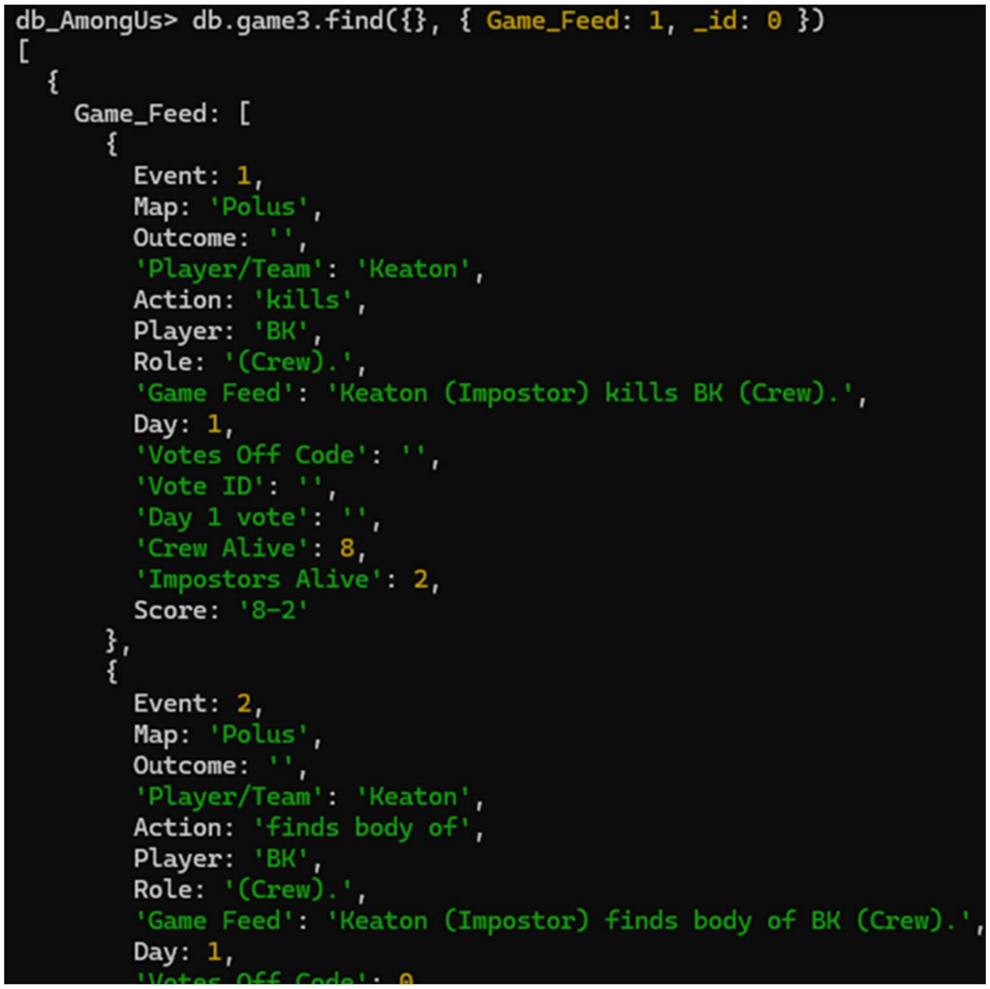
db.game3.insert(db.AmongUs.find({'game':'3'}).toArray())



```
db_AmongUs> db.game3.insert(db.AmongUs.find({'game':'3'}).toArray())
DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('66392608cc0e7672690999fc') }
}
db_AmongUs> db.game3.find()
[
  {
    _id: ObjectId('66392608cc0e7672690999fc'),
    game: '3',
    Game_Feed: [
      {
        Event: 1,
        Map: 'Polus',
        Outcome: '',
        'Player/Team': 'Keaton',
        Action: 'kills',
        Player: 'BK',
        Role: '(Crew).',
        'Game Feed': 'Keaton (Impostor) kills BK (Crew).',
        Day: 1,
        'Votes Off Code': '',
        'Vote ID': '',
        'Day 1 vote': '',
        'Crew Alive': 8,
        'Impostors Alive': 2,
        Score: '8-2'
      },
      {
        Event: 2,
        Map: 'Polus',
        Outcome: '',
        'Player/Team': 'Keaton',
        Action: 'finds body of',
        Player: 'BK',
        Role: '(Crew).',
        'Game Feed': 'Keaton (Impostor) finds body of BK (Crew).',
        Day: 1,
        'Votes Off Code': 0,
```

2.1. Show the Game Feed data specifically for game 3 in the newly created collection.

db.game3.find({},
{ Game_Feed: 1, _id: 0 })

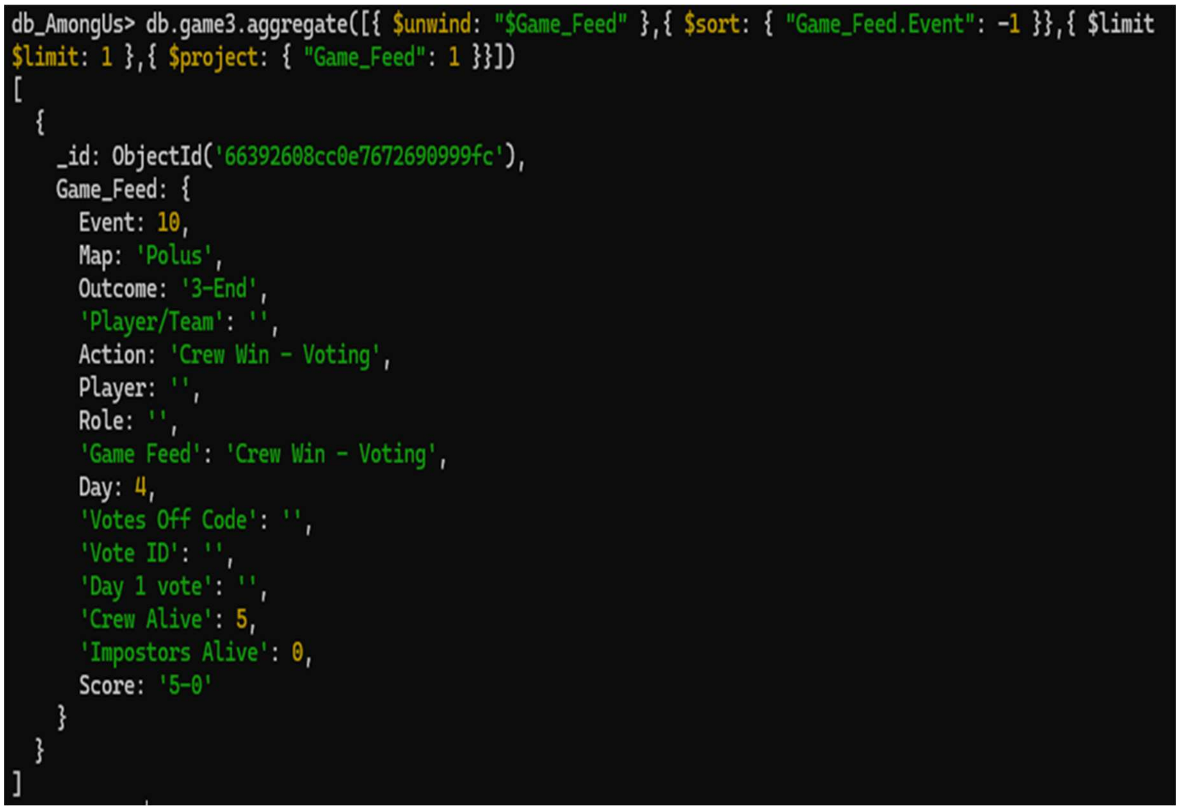


```
db_AmongUs> db.game3.find({}, { Game_Feed: 1, _id: 0 })
[
  {
    Game_Feed: [
      {
        Event: 1,
        Map: 'Polus',
        Outcome: '',
        'Player/Team': 'Keaton',
        Action: 'kills',
        Player: 'BK',
        Role: '(Crew).',
        'Game Feed': 'Keaton (Impostor) kills BK (Crew).',
        Day: 1,
        'Votes Off Code': '',
        'Vote ID': '',
        'Day 1 vote': '',
        'Crew Alive': 8,
        'Impostors Alive': 2,
        Score: '8-2'
      },
      {
        Event: 2,
        Map: 'Polus',
        Outcome: '',
        'Player/Team': 'Keaton',
        Action: 'finds body of',
        Player: 'BK',
        Role: '(Crew).',
        'Game Feed': 'Keaton (Impostor) finds body of BK (Crew).',
        Day: 1,
```

This MongoDB query searches the "game3" collection and retrieves all documents from the "game3" collection, but only the "Game_Feed" field will be included in the results; the "_id" field will be omitted.

2.2. Show the most recent event that occurred in game 3.

db.game3.aggregate([
{ $unwind: "$Game_Feed" },
{ $sort: { "Game_Feed.Event": -1 }},
{ $limit: 1 },
{ $project: { "Game_Feed": 1 }}])



```
db_AmongUs> db.game3.aggregate([{ $unwind: "$Game_Feed" },{ $sort: { "Game_Feed.Event": -1 }},{ $limit
$limit: 1 },{ $project: { "Game_Feed": 1 }}])
[
  {
    _id: ObjectId('66392608cc0e7672690999fc'),
    Game_Feed: {
      Event: 10,
      Map: 'Polus',
      Outcome: '3-End',
      'Player/Team': '',
      Action: 'Crew Win - Voting',
      Player: '',
      Role: '',
      'Game Feed': 'Crew Win - Voting',
      Day: 4,
      'Votes Off Code': '',
      'Vote ID': '',
      'Day 1 vote': '',
      'Crew Alive': 5,
      'Impostors Alive': 0,
      Score: '5-0'
    }
  }
]
```

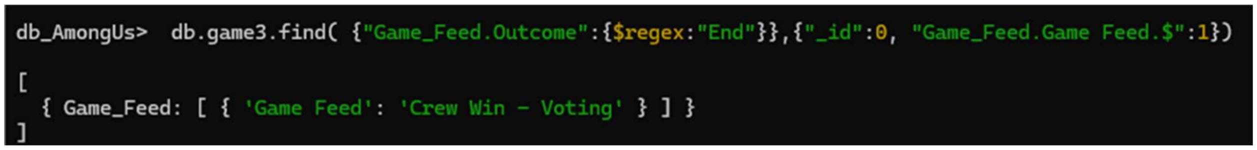
The MongoDB aggregation pipeline performs below operations:

$unwind: Splits the array field "Game_Feed" into separate documents for each element.
$sort: Arranges the documents based on the descending order of the "Event" field within "Game_Feed".
$limit: Limits the result to only one document (the one with the highest "Event").
$project: Projects or includes only the "Game_Feed" field in the output.

Result: Event 10

2.3. Determine the winner of game 3

db.game3.find(
{"Game_Feed.Outcome":{$regex:"End"}},
{"_id":0, "Game_Feed.Game Feed.$":1})



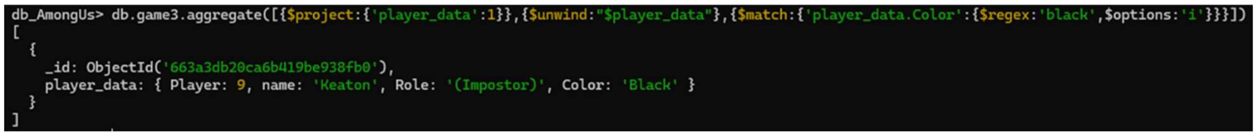
```
db_AmongUs>  db.game3.find( {"Game_Feed.Outcome":{$regex:"End"}},{"_id":0, "Game_Feed.Game Feed.$":1})

[
  { Game_Feed: [ { 'Game Feed': 'Crew Win - Voting' } ] }
]
```

Result: Crew

This query returns documents from the game3 collection where Game_Feed.Outcome contains "End", and for each matching document, it will include only the first element of the Game_Feed array within the Game Feed field, excluding the _id field.

2.4. Identify the player who chose the black color in game 3 and whether they were a crew member or impostor

db.game3.aggregate([
{$project:{'player_data':1}},
{$unwind:"$player_data"},
{$match:{'player_data.Color':{$regex:'black',$options:'i'}}}])



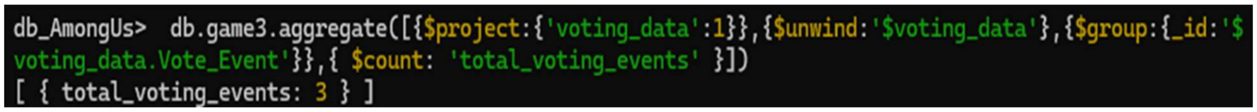
```
db_AmongUs> db.game3.aggregate([{$project:{'player_data':1}},{$unwind:"$player_data"},{$match:{'player_data.Color':{$regex:'black',$options:'i'}}}])
[
  {
    _id: ObjectId('663a3db20ca6b419be938fb0'),
    player_data: { Player: 9, name: 'Keaton', Role: '(Impostor)', Color: 'Black' }
  }
]
```

Result-
Name: Keaton,
Role: Impostor

The pipeline first extracts **player_data** from game3, then separates each array element into its own document, and finally filters to keep only documents where **player_data.Color** contains "black", ignoring the case.

2.5. Count the number of voting events that took place in game 3.

db.game3.aggregate([
{$project:{'voting_data':1}},
{$unwind:'$voting_data'},
{$group:{_id:'$voting_data.Vote_Event'}},{ $count: 'total_voting_events' }])



```
db_AmongUs>  db.game3.aggregate([{$project:{'voting_data':1}},{$unwind:'$voting_data'},{$group:{_id:'$
voting_data.Vote_Event'}},{ $count: 'total_voting_events' }])
[ { total_voting_events: 3 } ]
```

Result - Vote_Events:3

This pipeline extracts **voting_data** from each document, separates each array element into its own document, groups documents by the Vote Event, and finally counts the distinct Vote Events to get the total number of voting events.

2.6. If you were redesigning this database for easier querying, describe the changes you would make and explain your reasoning.

Optimizing the data structure and database design can significantly improve performance and reduce complexity. Here are some strategies that can be implemented:

Normalization: Breaking down data into smaller, more manageable tables and establishing relationships between them can help reduce redundancy and improve data integrity. This can involve creating separate collections for distinct entities like players, games, votes, etc. Creating separate collections for player data and voting data and then referencing them in the game feed through $LOOKUP can simplify queries and improve database organization. This approach adheres to the principles of data normalization and reduces data duplication.

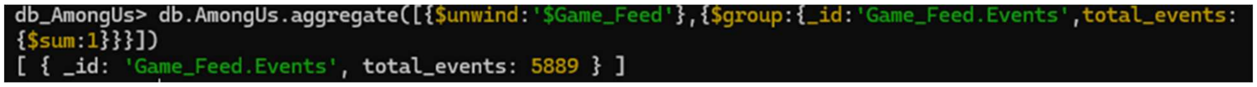
Indexing: Creating indexes on commonly queried fields can speed up search and retrieval operations, leading to faster response times. Indexes can be applied to fields frequently used in filters, sorts or in fields that are involved in equality matches.

Schema Redesign: Analyzing and redesigning the schema to minimize nested fields and eliminate redundant data can streamline queries and reduce the need for expensive operations like $unwind.

Task 3: Overall Aggregation

3.1. Calculate the total number of events recorded in this collection across all games.

db.AmongUs.aggregate([
{$unwind:'$Game_Feed’},
{$group:{_id:'Game_Feed.Events’, total_events:{$sum:1}}}])



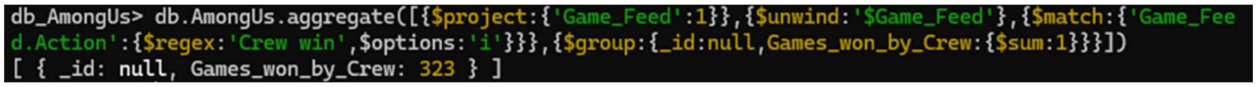
```
db_AmongUs> db.AmongUs.aggregate([{$unwind:'$Game_Feed'},{$group:{_id:'Game_Feed.Events',total_events:
{$sum:1}}}])
[ { _id: 'Game_Feed.Events', total_events: 5889 } ]
```

Result- **Total events:5889**

The pipeline separates each Game_Feed element into its own document, then groups documents by their Game_Feed.Events field and calculates the total count of events for each group

3.2. Compare the crew's wins to the impostors' wins and provide the counts.

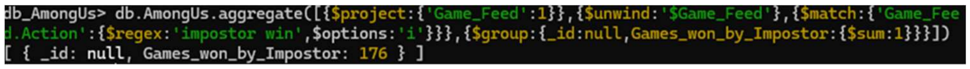
db.AmongUs.aggregate([
{$project:{'Game_Feed':1}},
{$unwind:'$Game_Feed’},
{$match:{'Game_Feed.Action':{$regex:’ Crew win',$options:'i’ }}},
{$group:{_id:null,Games_won_by_Crew: {$sum:1}}}])



```
db_AmongUs> db.AmongUs.aggregate([{$project:{'Game_Feed':1}},{$unwind:'$Game_Feed'},{$match:{'Game_Fee
d.Action':{$regex:'Crew win',$options:'i'}}},{$group:{_id:null,Games_won_by_Crew:{$sum:1}}}])
[ { _id: null, Games_won_by_Crew: 323 } ]
```

Crew Won: **323 times**

db.AmongUs.aggregate([
{$project:{'Game_Feed':1}},
{$unwind:'$Game_Feed’},
{$match:{'Game_Feed.Action':{$regex: 'impostor win',$options:'i'}}},
{$group :{_id:null, Games_won_by_Impostor: {$sum:1}}}])



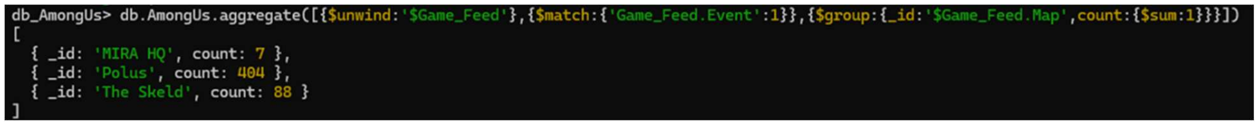
```
db_AmongUs> db.AmongUs.aggregate([{$project:{'Game_Feed':1}},{$unwind:'$Game_Feed'},{$match:{'Game_Fee
d.Action':{$regex:'impostor win',$options:'i'}}},{$group:{_id:null,Games_won_by_Impostor:{$sum:1}}}])
[ { _id: null, Games_won_by_Impostor: 176 } ]
```

Imposter Won: 176 times

$project: Selects the Game_Feed field from the document.
$unwind: Deconstructs the Game_Feed array from each document into separate documents, one for each array element.
$match: Filters documents to include only those where the Game_Feed.Action field contains the substring "Crew win /Impostor win".
$group: Groups the documents (since _id: null) and calculates the count of documents where Crew/Impostor wins by summing them up.

3.3. List the maps played and the total number of games on each map.

db.AmongUs.aggregate([
{$unwind:"$Game_Feed"},
{$match:{‘Game_Feed.Event’:1}},
{$group:{_id:"$Game_Feed.Map", count:{$sum:1}}}])



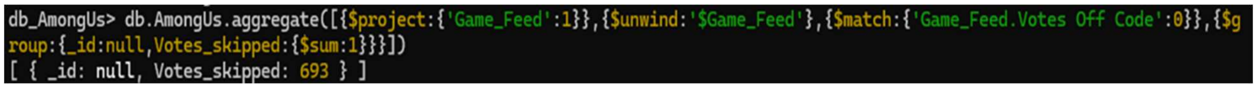
```
db_AmongUs> db.AmongUs.aggregate([{$unwind:'$Game_Feed'},{$match:{'Game_Feed.Event':1}},{$group:{_id:'$Game_Feed.Map',count:{$sum:1}}}])
[
  { _id: 'MIRA HQ', count: 7 },
  { _id: 'Polus', count: 404 },
  { _id: 'The Skeld', count: 88 }
]
```

Result : 'MIRA HQ’ count: 7 'Polus’ count: 404 'The Skeld’ count: 88

Here, pipeline separates each Game_Feed element into its own document, filters to include only documents where Game_Feed.Event is 1, and then groups by the Game_Feed.Map field to count occurrences of each map in the filtered documents.

3.4. Determine the total instances of crew members skipping a vote across all games.

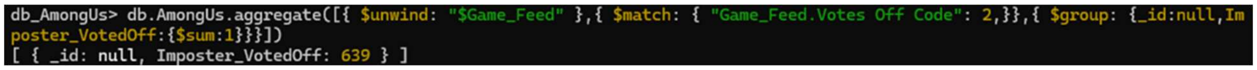
db.AmongUs.aggregate([
{ $unwind: "$Game_Feed" },
{ $match: { "Game_Feed.Votes Off Code": 0,}},
{ $group: {_id:null, Votes_skipped:{$sum:1}}}])


```
db_AmongUs> db.AmongUs.aggregate([{$project:{'Game_Feed':1}},{$unwind:'$Game_Feed'},{$match:{'Game_Feed.Votes Off Code':0}},{$g
roup:{_id:null,Votes_skipped:{$sum:1}}}])
[ { _id: null, Votes_skipped: 693 } ]
```
Result: 693

This pipeline separates each Game_Feed element into its own document, filters to include only documents where the vote off code is 2 (representing imposters voted off), and then counts the occurrences of imposters being voted off across all documents.

3.5. Calculate the total occurrences of crew members voting against imposters across all matches

```
db.AmongUs.aggregate([
{ $unwind: "$Game_Feed" },
{ $match: { "Game_Feed.Votes Off Code": 2,}},
{ $group: {_id:null, Imposter_VotedOff:{$sum:1}}}])
```


```
db_AmongUs> db.AmongUs.aggregate([{ $unwind: "$Game_Feed" },{ $match: { "Game_Feed.Votes Off Code": 2,}},{ $group: {_id:null,Im
poster_VotedOff:{$sum:1}}}])
[ { _id: null, Imposter_VotedOff: 639 } ]
```
Result: 639 times

This pipeline separates each Game_Feed element into its own document, filters to include only documents where the vote off code is 2 (representing imposters voted off), and then counts the occurrences of imposters being voted off across all documents.

3.6. Share your opinion on whether the game is more or less challenging for impostors, supported by insights from the data.

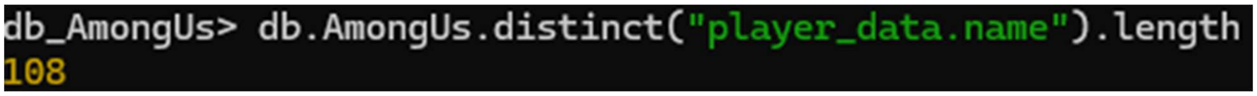
- Analyzing win-loss ratios, kill counts, and voting patterns is essential to assess whether impostors were disadvantaged or not.
- With a total of 323 wins for the crew and 176 wins for the Imposters, these statistics suggest that the game leans towards being easier for the crew and hints at a greater challenge faced by impostors.
- Crew members cast 639 votes against impostors. This frequent occurrence, coupled with the crew's higher win rate, implies that crew members excel at spotting and voting out impostors

In conclusion, the data strongly indicates that the game presents a great challenge for impostors and shows the effectiveness of crew members in maintaining a competitive edge within the game environment.

Task 4 – Player level aggregation

4.1. Find the count of unique players in the dataset.

db.AmongUs.distinct("player_data.name").length



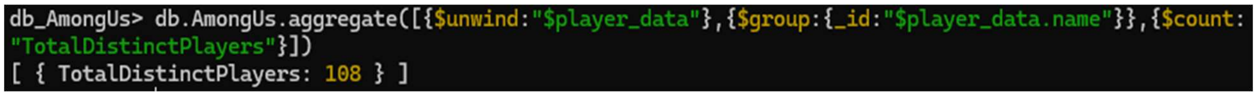
```
db_AmongUs> db.AmongUs.distinct("player_data.name").length
108
```

Result: 108 Players

The query returns the number of unique player names recorded in the player_data.name field across all documents in the AmongUs collection.

OR

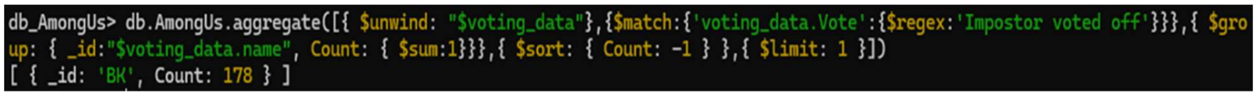
```
db.AmongUs.aggregate([
{$unwind:"$player_data"},
{$group:{_id:"$player_data.name"}},
{$count:“TotalDistinctPlayers"}])
```



```
db_AmongUs> db.AmongUs.aggregate([{$unwind:"$player_data"},{$group:{_id:"$player_data.name"}},{$count:
"TotalDistinctPlayers"}])
[ { TotalDistinctPlayers: 108 } ]
```

4.2. Identify the player considered the best crew member.

db.AmongUs.aggregate([
{ $unwind: "$voting_data"},
{ $group: { _id:"$voting_data.name", Count: { $sum:1}}},
{ $sort: { Count: -1 } },{ $limit: 1 }])



```
db_AmongUs> db.AmongUs.aggregate([{ $unwind: "$voting_data"},{$match:{'voting_data.Vote':{$regex:'Impostor voted off'}}},{ $gro
up: { _id:"$voting_data.name", Count: { $sum:1}}},{ $sort: { Count: -1 } },{ $limit: 1 }])
[ { _id: 'BK', Count: 178 } ]
```

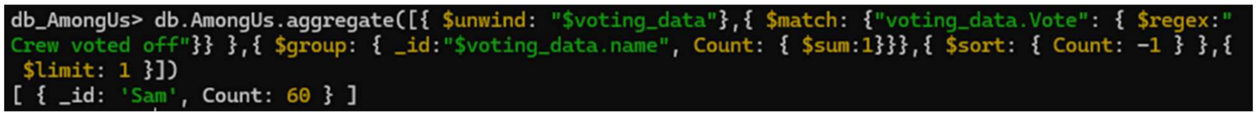
Result: BK

The aggregation pipeline on the **AmongUs** collection performs the following operations:

$unwind: Deconstructs the **voting_data** array from each document into separate documents, one for each array element.
$group: Groups the documents by the **voting_data.name** field and calculates the count of documents in each group (**Count**) by summing them up.
$sort: Sorts the grouped data in descending order based on the **Count** field, which represents the number of occurrences of each player's name.
$limit: Limits the output to only the first document after sorting, effectively retrieving the player with the highest count of occurrences in the **voting_data.name** field.

4.3. Identify the player regarded as the least effective crew member.

db.AmongUs.aggregate([
{ $unwind: "$voting_data"},
{ $match: {"voting_data.Vote": { $regex:"Crew voted off"}}},
{ $group: { _id:"$voting_data.name", Count: { $sum:1}}},
{ $sort: { Count: -1 }},
{ $limit: 1 }])



```
db_AmongUs> db.AmongUs.aggregate([{ $unwind: "$voting_data"},{ $match: {"voting_data.Vote": { $regex:"
Crew voted off"}} },{ $group: { _id:"$voting_data.name", Count: { $sum:1}}},{ $sort: { Count: -1 } },{
 $limit: 1 }])
[ { _id: 'Sam', Count: 60 } ]
```

Result: Sam

The aggregation pipeline on the **AmongUs** collection performs the following operations:

$unwind: Deconstructs the **voting_data** array from each document into separate documents, one for each array element.
$match: Filters documents to include only those where the **voting_data.Vote** field contains the substring "Crew voted off" using a regex pattern.
$group: Groups the documents by the **voting_data.name** field and calculates the count of documents in each group (**Count**) by summing them up.
$sort: Sorts the grouped data in descending order based on the **Count** field, which represents the number of occurrences of each player's name where they voted for the crew to be voted off.
$limit: Limits the output to only the first document after sorting, effectively retrieving the player who has voted for the crew to be voted off the most times.