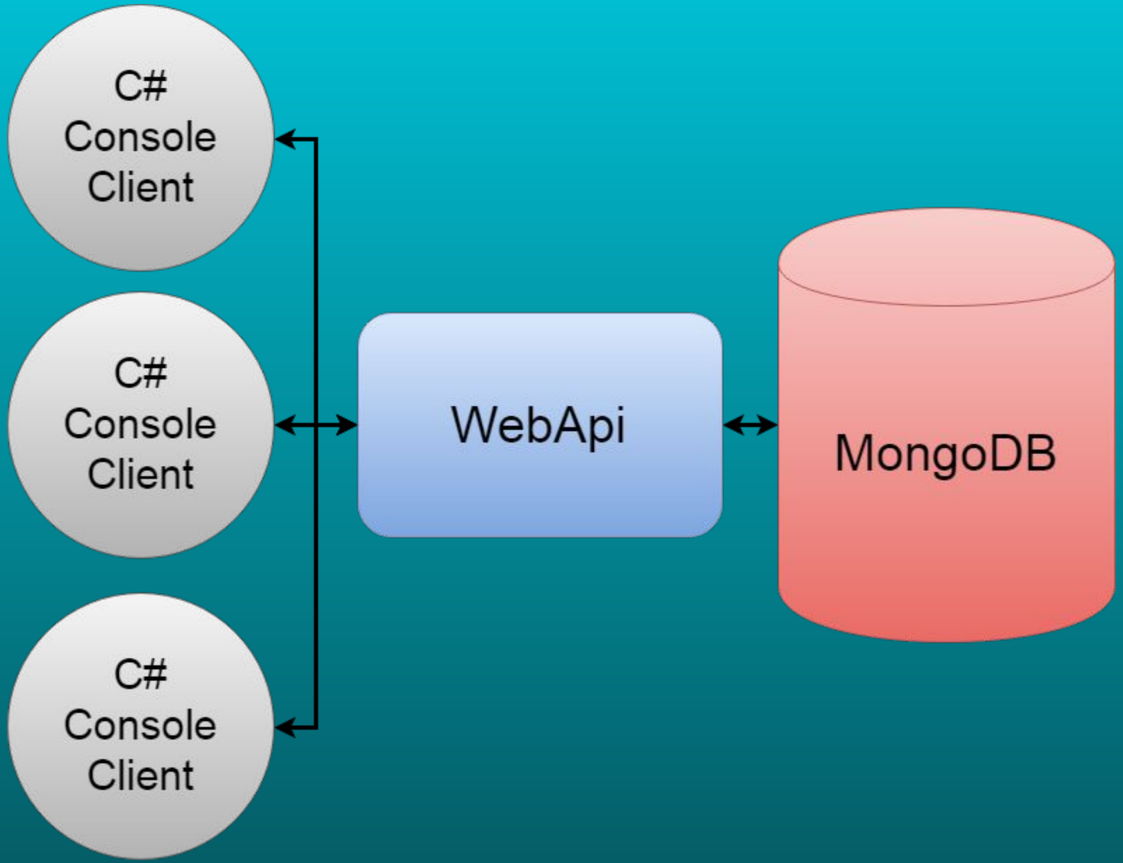
Connect4 ~ 9x9

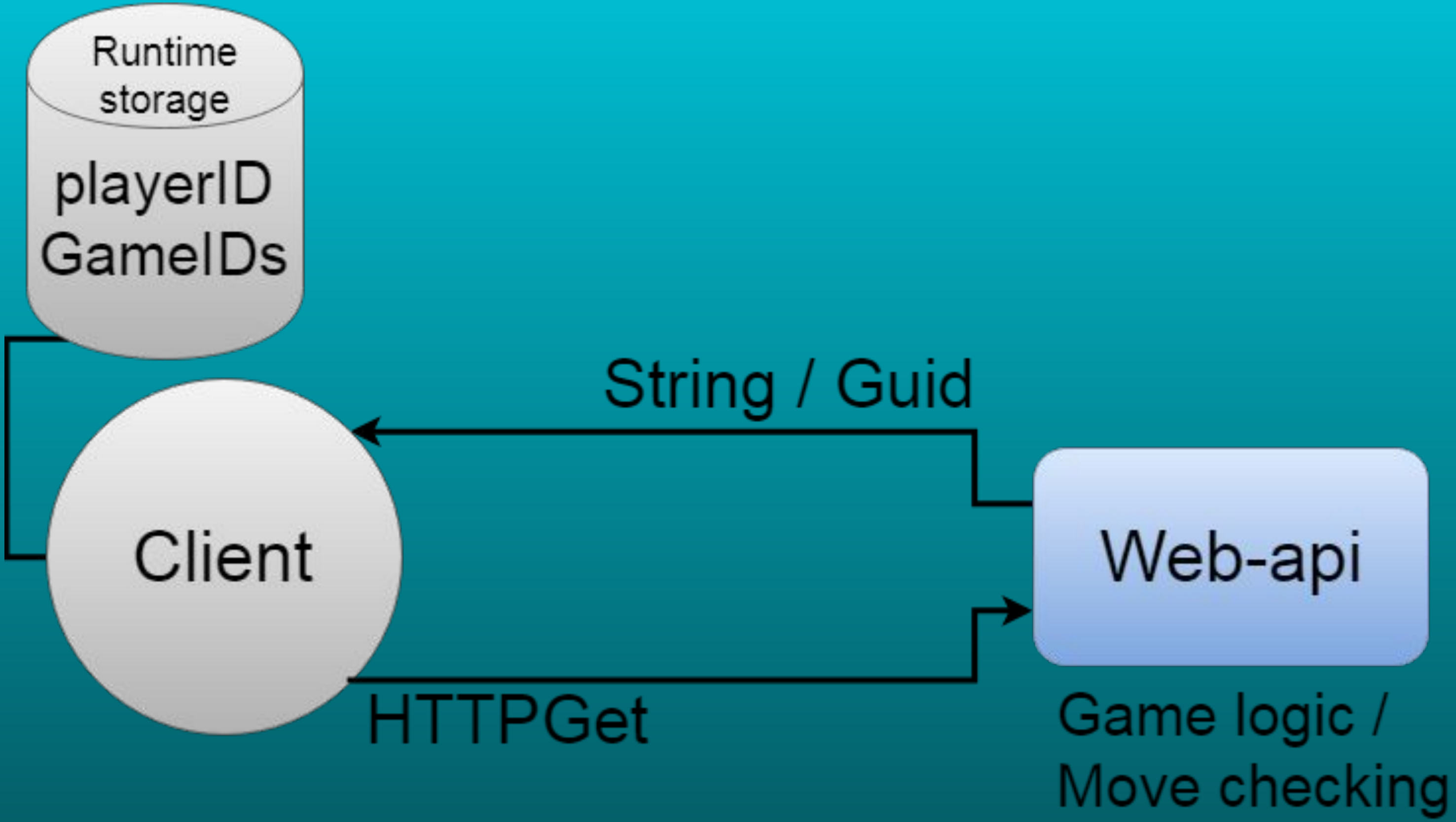
Sakari Helokunnas

Content

Basic structure



Implementation



Implementation (cont.)

-Client library: Microsoft.AspNet.WebApi.Client

-Moves are passed as query parameters (eg. ?move={move})

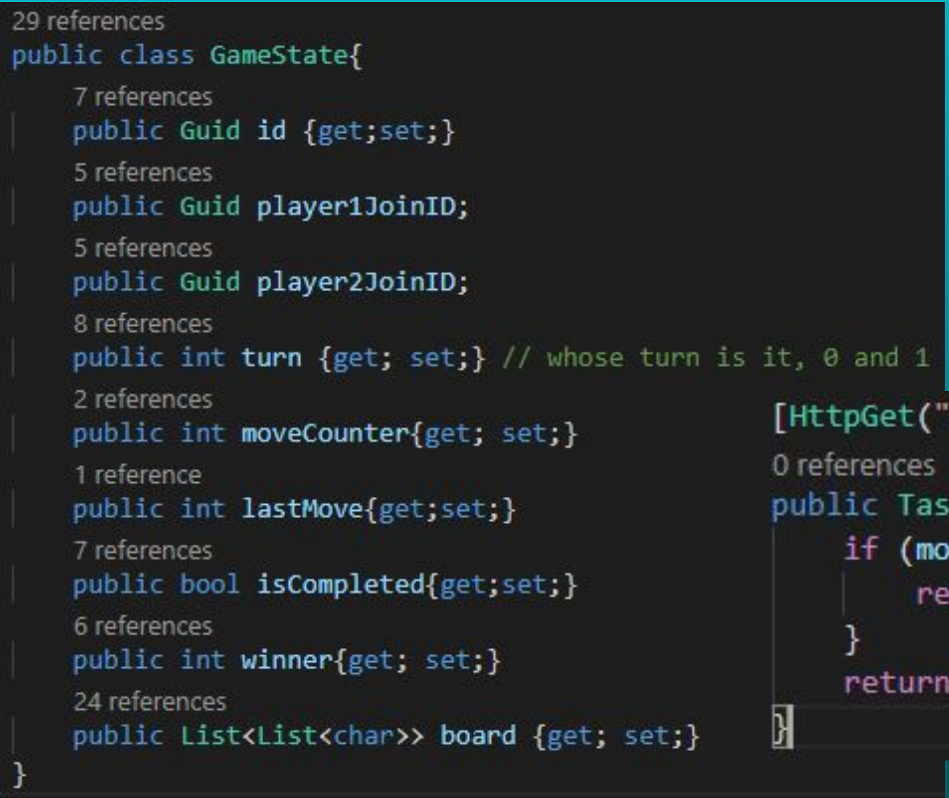
-Recurring queries while "waiting" for new information

-Client listens to ESC press in another thread

    -Prevent softlocking during waiting

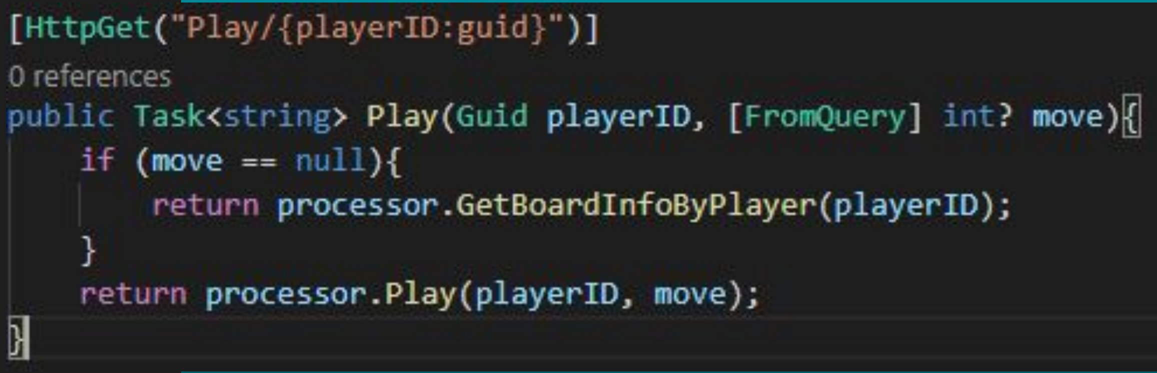
Implementation (cont.)

Database object:



```
29 references
public class GameState{
    7 references
    public Guid id {get;set;}
    5 references
    public Guid player1JoinID;
    5 references
    public Guid player2JoinID;
    8 references
    public int turn {get; set;} // whose turn is it, 0 and 1
    2 references
    public int moveCounter{get; set;}
    1 reference
    public int lastMove{get;set;}
    7 references
    public bool isCompleted{get;set;}
    6 references
    public int winner{get; set;}
    24 references
    public List<List<char>> board {get; set;}
}
```

Controller method example:



```
[HttpGet("Play/{playerID:guid}")]
0 references
public Task<string> Play(Guid playerID, [FromQuery] int? move){
    if (move == null){
        return processor.GetBoardInfoByPlayer(playerID);
    }
    return processor.Play(playerID, move);
}
```

Demo

Challenges

-Recurring queries VERSUS Waiting in WebApi

-JSon wrapper in client

-Building and testing client and WebApi at the same time

-Multithreading and Console

-Time

Defects

-Some logic at Repository level

-Aggregate(ing) and returning anonymous types

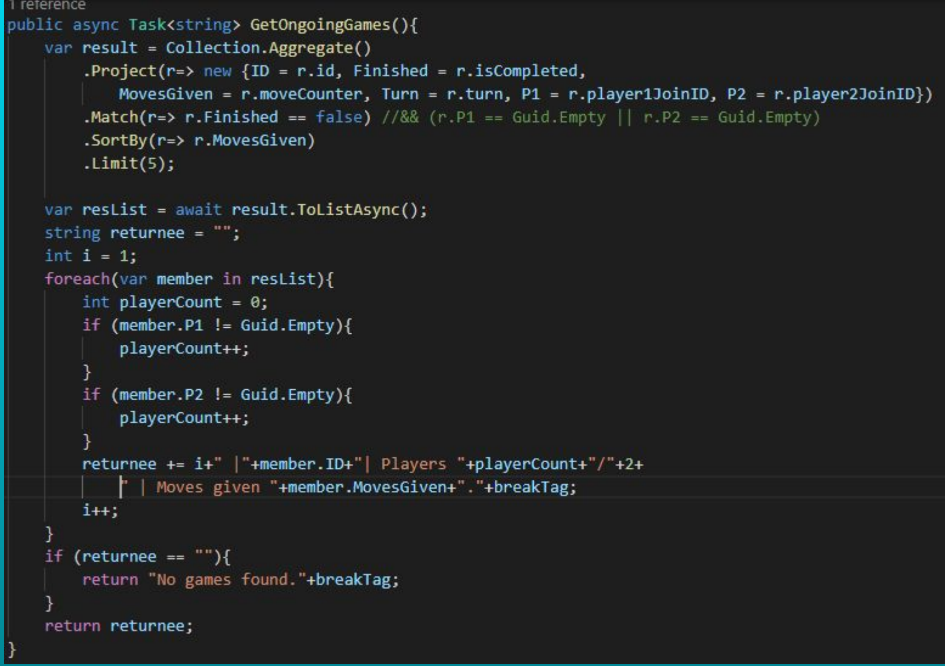
-No join messages

-Relevant only with remote api

-Insecure connections

-Sending playerID with each Get request

-No data/error logging



```
1 reference
public async Task<string> GetOngoingGames(){
    var result = Collection.Aggregate()
        .Project(r=> new {ID = r.id, Finished = r.isCompleted,
            MovesGiven = r.moveCounter, Turn = r.turn, P1 = r.player1JoinID, P2 = r.player2JoinID})
        .Match(r=> r.Finished == false) //&& (r.P1 == Guid.Empty || r.P2 == Guid.Empty)
        .SortBy(r=> r.MovesGiven)
        .Limit(5);

    var resList = await result.ToListAsync();
    string returnee = "";
    int i = 1;
    foreach(var member in resList){
        int playerCount = 0;
        if (member.P1 != Guid.Empty){
            playerCount++;
        }
        if (member.P2 != Guid.Empty){
            playerCount++;
        }
        returnee += i+" |"+member.ID+"| Players "+playerCount+"/"+2+
            " | Moves given "+member.MovesGiven+"."+breakTag;
        i++;
    }
    if (returnee == ""){
        return "No games found."+breakTag;
    }
    return returnee;
}
```

Thank you

Any questions?