WARSAW UNIVERSITY OF TECHNOLOGY

THE FACULTY OF ELECTRONICS AND INFORMATION TECHNOLOGY

Distributed Computing and Systems:

Project documentation

Game “Mensch ärgere Dich nicht”

Team members: Dora Doljanin, Filip Pavičić

Supervisor: Mr. Oleszkiewicz

Warsaw, June 2022.

1. Description of the task

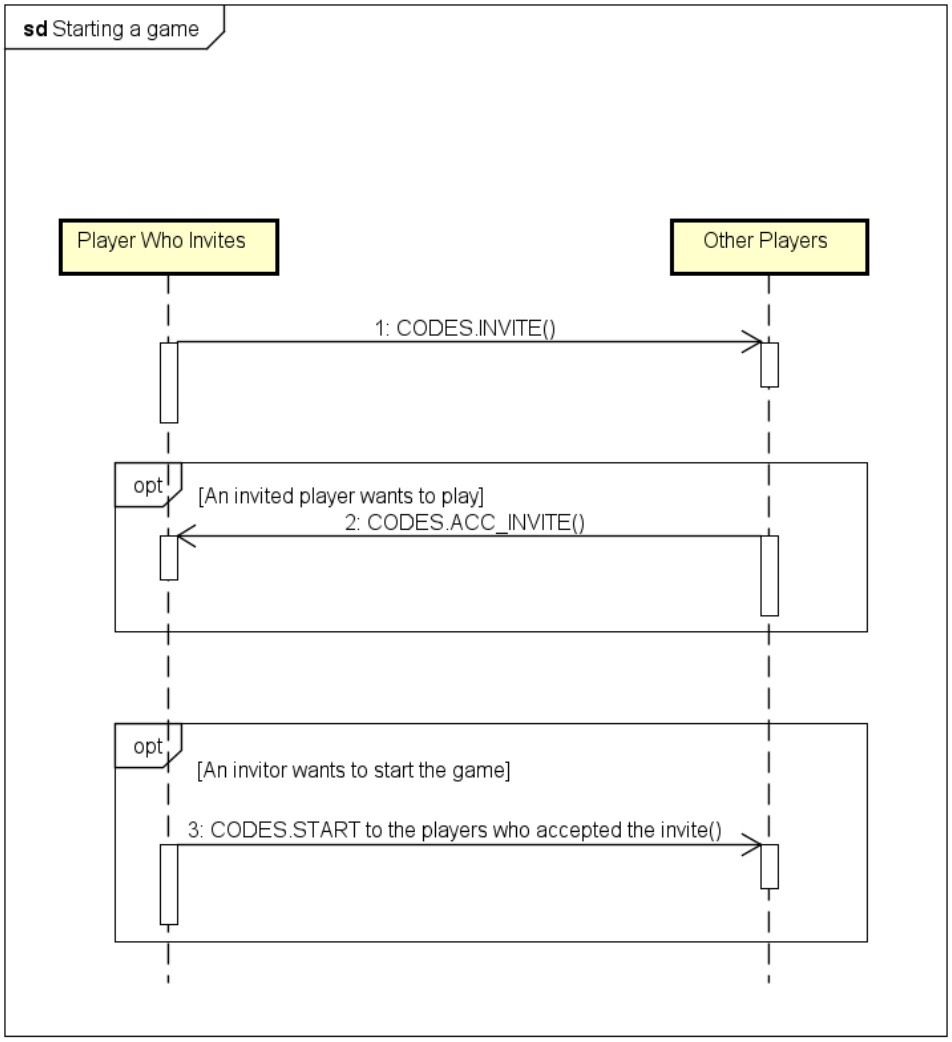
Our objective is to implement the game “Mensch ärgere Dich nicht” as a distributed system. The game can be played by up to 4 people at the same time. The game starts when everyone willing to play the game is ready. In a situation where one of the players disconnects from the game, the pieces of that player should remain on the board. The disconnected user should be able to return to the game at any time. When the user is disconnected, its movements are skipped, and the rest of the players continue playing the game.

2. General assumptions and system architecture

We implement our solution to the proposed task in two different languages: Python and Java. Both implementations work on any operational system. We test them in Windows OS and Linux OS. The players communicate their moves with each other via messages, which is implemented using peer-to-peer UDP communication. Each player client has a thread which constantly listens for incoming messages. Furthermore, each player client handles only their own moves and informs the other players about them. If a player exits the game, that does not affect other players, who continue to play the game.

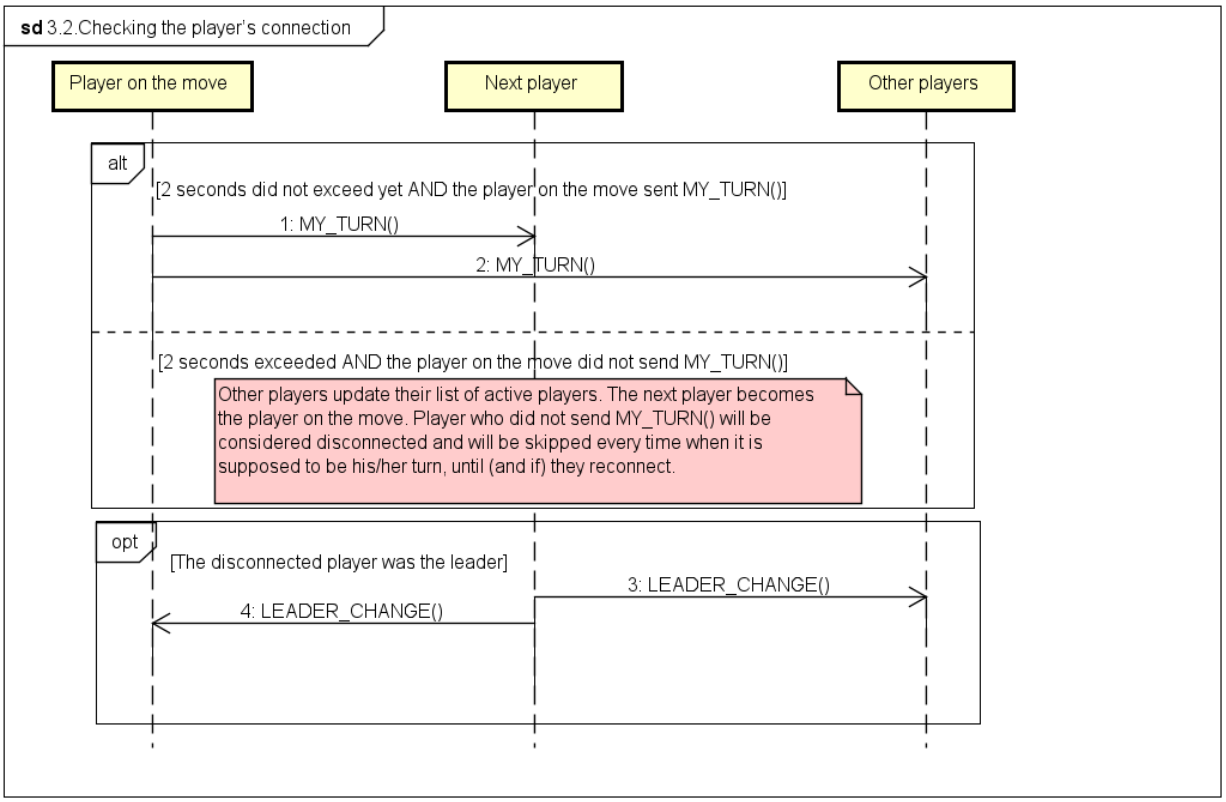
3. Use cases

a. Starting the game



The above diagram represents the use case of starting the game. We implemented this action the same way we designed it in the project concepts. The player who invites the others becomes the leader of the game.

b. Checking the player's connection

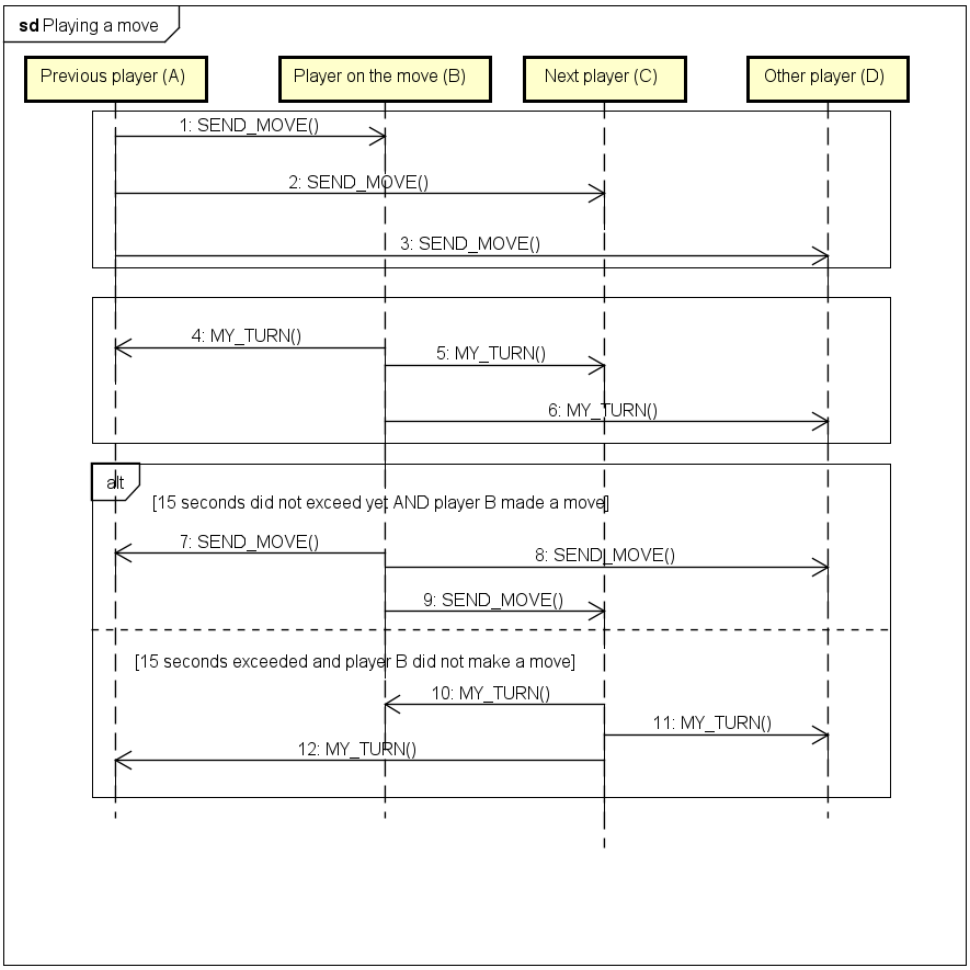


The diagram demonstrates our initial idea of implementing the checking of players connection. However, while developing our application, we decided to simplify the implementation of this task.

If one of the players disconnects from the game, other players receive information about that event via a message and they simply remove that player from the list of players. They do not wait for timeout of the disconnected player's response; they just skip his turn and continue to play without him.

An example of this action will be demonstrated in the following chapters.

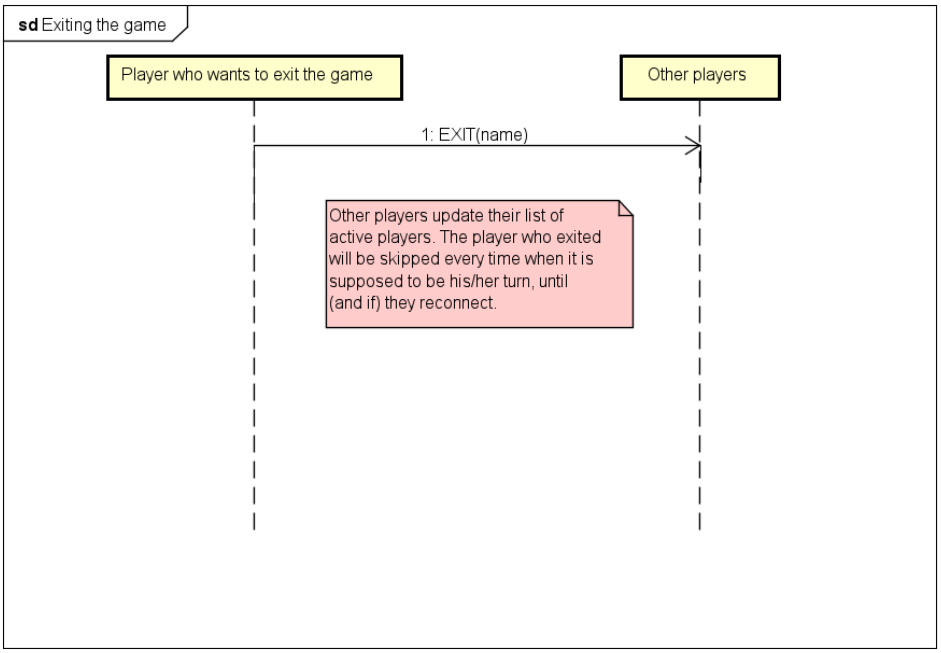
c. Playing a move



The diagram above illustrates our initial idea of playing a move. However, during the implementation process, we adjusted this idea to make it more efficient.

After one player makes the move, the leader looks at the list of players in the game and determines who is next in the list after the current player. Then, the leader broadcasts to everyone who is next on the move, so everyone knows who is playing next.

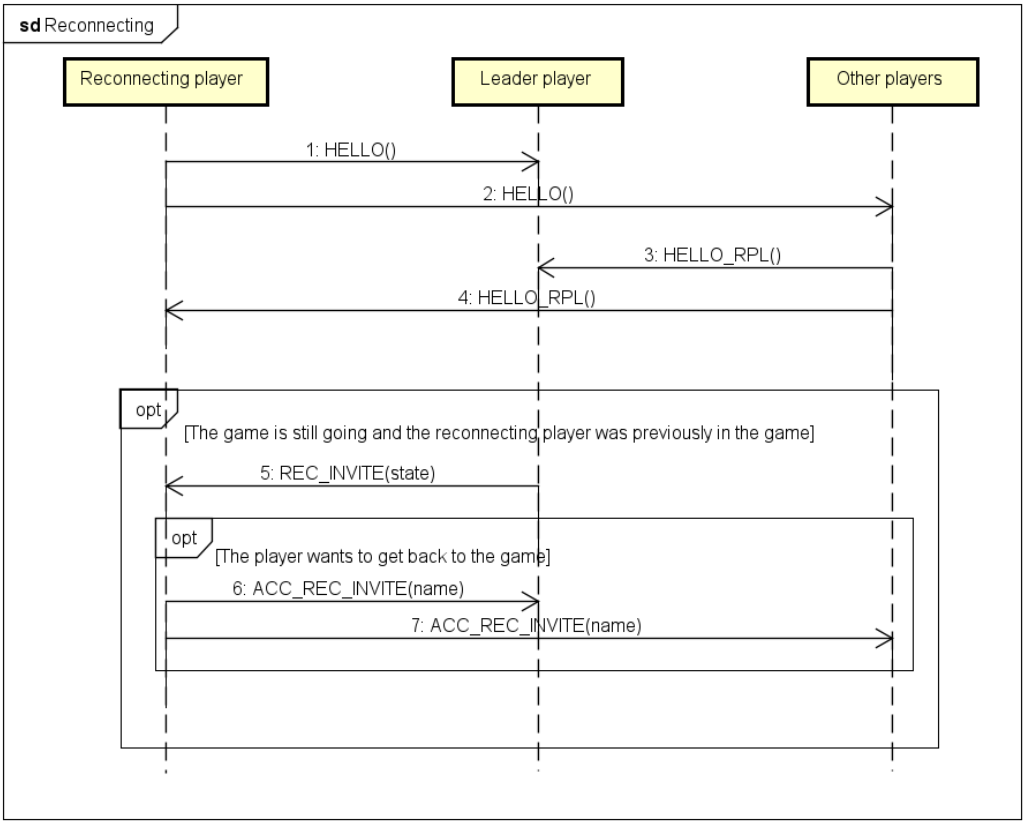
d. Exiting the game



The diagram shows the use case of exiting the game in the way that we initially designed it and the way we implemented it.

The action is very simple; the player who wants to exit the game just sends the “exit” message to all other players. That way, the player informs other players about leaving the game and he is deleted from the list of players.

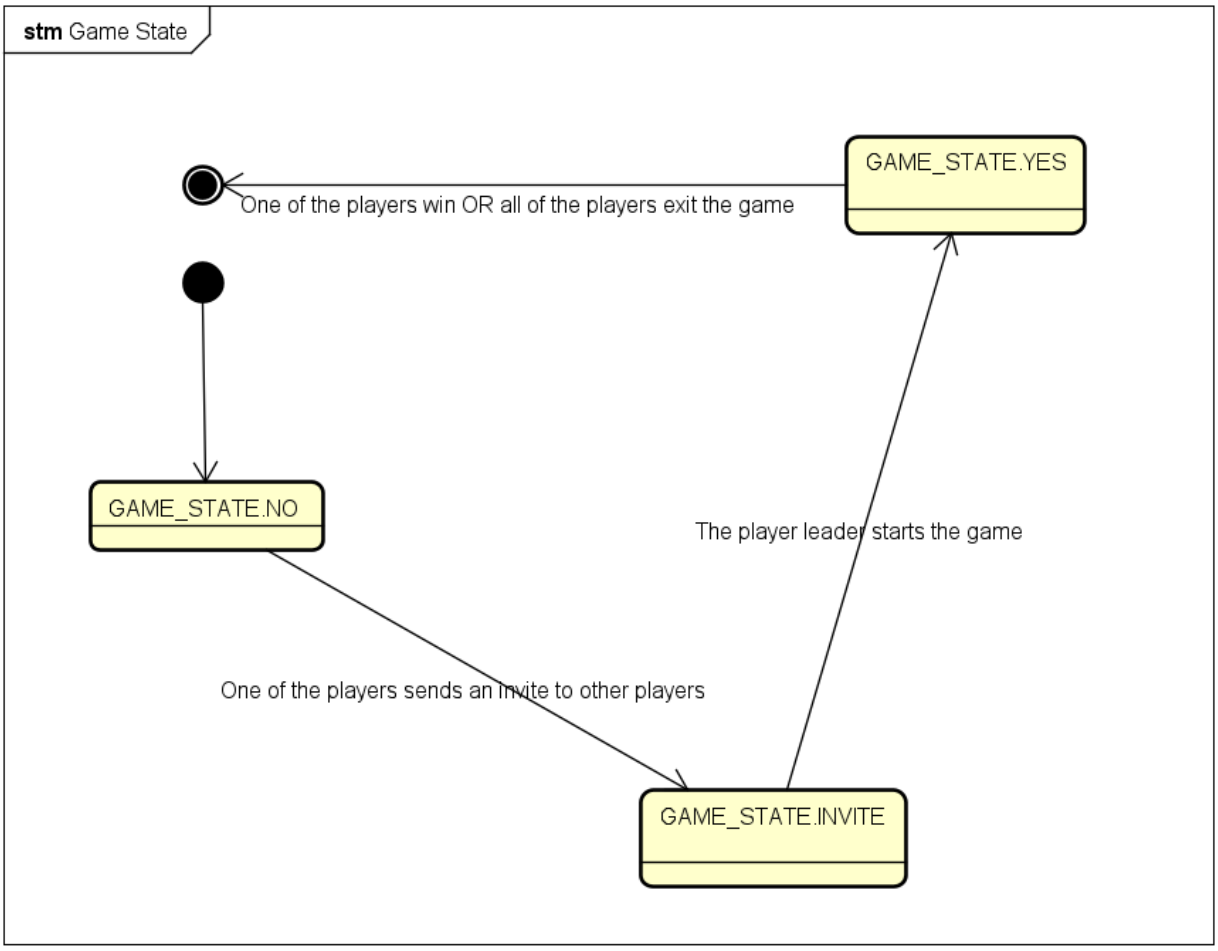
e. Reconnecting



The diagram above illustrates our initial plan of implementing player's returning to the game. However, while developing our application we changed the initial idea of that action in the following way:

If a player who previously exited the game wants to return to the game, he just runs the application and enters the game again with the same username as before. Then, he is added at the end of the list of players and continues playing the game when it is his turn again, according to the order in the list of players. All his previous moves and pieces' locations are remembered by the application and restored when he comes back. He does not have to wait for the leader's invite back to the game. The leader and the other users are simply informed about his return and continue playing the game with him.

4. Game state



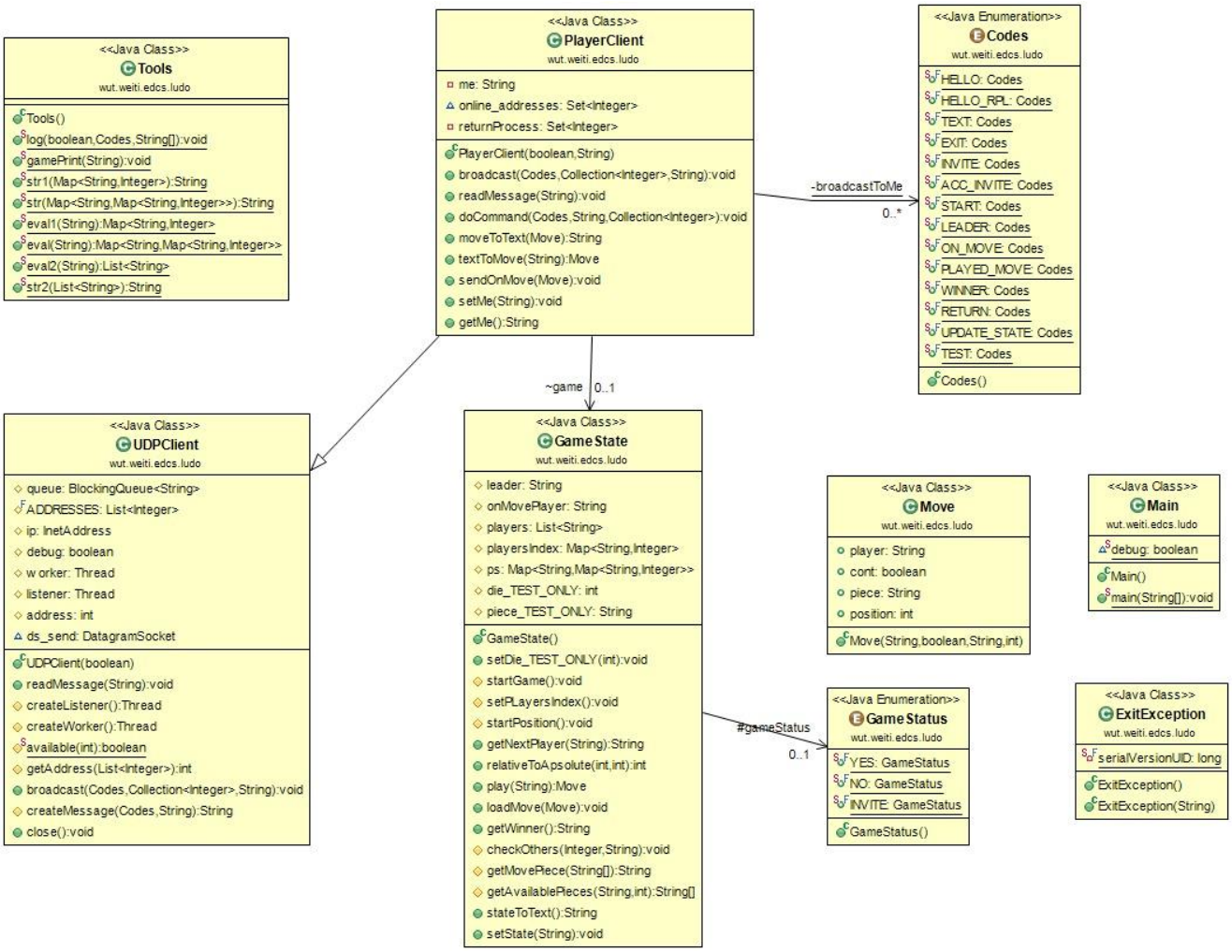
The game state diagram shows the state of the game as we initially designed them implemented them.

There are three game states:

- **YES** – after starting the game
- **NO** – before the invite
- **INVITE** – when one of the players has invited other players but has not started the game yet

5. Class diagram

The following diagram shows our class diagram generated by Eclipse IDE.



6. Report of tests

We have adjusted and expanded our test plan with respect to the initial test plan and we have decided to test the following scenarios:

- Disconnecting of a **non-leader** player who is currently **not on the move** during the game
- Disconnecting of a **non-leader** player who is currently **on the move** during the game
- Disconnecting of a **leader** player who is currently **not on the move** during the game
- Disconnecting of a **leader** player who is currently **on the move** during the game
- **More than one** player disconnecting from the game
- Player's **return** to the game
- Playing **forbidden** moves
- Retrieving **available pieces** for the move
- **Initializing** the board
- Calculating the **absolute position** from the player's **relative position**
- Casting other players' pieces out of the game ("**killing**" other players' pieces)

We created Unit tests for various scenarios to test if our application works correctly.

Firstly, we define 4 players: Filip, Dora, Ana, and Hari. We initialize a new game state, set the players, and start the game. We also create a dictionary which has all of the pieces set to position 0, which represents the place where the pieces are located before they enter the game.

In the test0, we simply check if the initial positions of all pieces at the beginning of the game are equal to those specified in the dictionary (all zeros).

```
from gameState import GameState

players = ["Filip","Dora","Ana","Hari"]
gs = GameState()
gs.setPLayers(players)
gs.startGame()
dic = { # relative positions for each player
    "Filip" : {"a":0,"b":0,"c":0,"d":0},
    "Dora" : {"a":0,"b":0,"c":0,"d":0},
    "Ana" : {"a":0,"b":0,"c":0,"d":0},
    "Hari" : {"a":0,"b":0,"c":0,"d":0},
}
def test0():
    assert gs.ps == dic, "Test0 Failed"
```

Then, in the test1, we check if the absolute position is calculated accurately from Filip's relative position of the piece.

```
def test1():
    relative = 0
    absolute = gs.relativeToApsolute(relative, gs.playersIndex["Filip"])
    assert absolute == 0, "Test1 Failed"
```

In the test2, test3, and test4, we check if the absolute positions are calculated accurately with respect to players' relative positions of their pieces.

```
def test2():
    relative = 1
    absolute = gs.relativeToApsolute(relative, gs.playersIndex["Filip"])
    assert absolute == 1, "Test2 Failed Filip"
    absolute = gs.relativeToApsolute(relative, gs.playersIndex["Dora"])
    assert absolute == 11, "Test2 Failed Dora"
    absolute = gs.relativeToApsolute(relative, gs.playersIndex["Ana"])
    assert absolute == 21, "Test2 Failed Ana"
    absolute = gs.relativeToApsolute(relative, gs.playersIndex["Hari"])
    assert absolute == 31, "Test2 Failed Hari"
```

```
def test3():
    relative = 15
    absolute = gs.relativeToApsolute(relative, gs.playersIndex["Filip"])
    assert absolute == 15, "Test3 Failed Filip"
    absolute = gs.relativeToApsolute(relative, gs.playersIndex["Dora"])
    assert absolute == 25, "Test3 Failed Dora"
    absolute = gs.relativeToApsolute(relative, gs.playersIndex["Ana"])
    assert absolute == 35, "Test3 Failed Ana"
    absolute = gs.relativeToApsolute(relative, gs.playersIndex["Hari"])
    assert absolute == 5, "Test3 Failed Hari"
```

```
def test4():
    relative = 41
    absolute = gs.relativeToApsolute(relative, gs.playersIndex["Filip"])
    assert absolute == 41, "Test4 Failed Filip"
    absolute = gs.relativeToApsolute(relative, gs.playersIndex["Dora"])
    assert absolute == 51, "Test4 Failed Dora"
    absolute = gs.relativeToApsolute(relative, gs.playersIndex["Ana"])
    assert absolute == 61, "Test4 Failed Ana"
    absolute = gs.relativeToApsolute(relative, gs.playersIndex["Hari"])
    assert absolute == 71, "Test4 Failed Hari"
```

In the test5, we check if the next player is determined correctly, according to the defined order: Filip, Dora, Ana, Hari, Filip, Dora, and so on.

```
def test5():
    nextPlayer = gs.getNextPlayer("Filip")
    assert nextPlayer == "Dora", "Test5 Failed Filip"
    nextPlayer = gs.getNextPlayer("Dora")
    assert nextPlayer == "Ana", "Test5 Failed Dora"
    nextPlayer = gs.getNextPlayer("Ana")
    assert nextPlayer == "Hari", "Test5 Failed Ana"
    nextPlayer = gs.getNextPlayer("Hari")
    assert nextPlayer == "Filip", "Test5 Failed Hari"
```

In the test6, we check if the method for getting available pieces retrieves the correct pieces for each of the players. We can see that all Filip's pieces are still at the position 0 (outside of the game), so when he gets a 6, all of them are available to be chosen for that move. Similarly, we can see that all Dora's pieces are still at the position 0 (outside of the game), but since she gets a 4, none of them are available to use for the move, because they can only get out with a 6. Ana's pieces "a" and "b" are in the game, while "c" and "d" are also in the game, but they have already entered the house. When Ana gets a 3, "a", "b", and "c" are available to move, but "d" is not because it does not have enough space in the house to move 3 steps forward. Finally, when Hari gets a 6, he can move the piece "a" on the position 0 and the piece "d", which is in the game, but he can not move pieces "b" and "c", since they are already inside the house and do not have 6 steps in front of them to move.

```
def test6():
    dic1 = { # relative positions for each player
    "Filip" : {"a":0,"b":0,"c":0,"d":0},
    "Dora" : {"a":0,"b":0,"c":0,"d":0},
    "Ana" : {"a":5,"b":22,"c":41,"d":43},
    "Hari" : {"a":0,"b":41,"c":42,"d":38},
    }
    gs.ps = dic1
    pieces = gs.getAvailablePieces("Filip", 6)
    assert pieces == ["a","b","c","d"], "Test6 Failed Filip"
    pieces = gs.getAvailablePieces("Dora", 4)
    assert pieces == [], "Test6 Failed Dora"
    pieces = gs.getAvailablePieces("Ana", 3)
    assert pieces == ["a","b","c"], "Test6 Failed Ana"
    pieces = gs.getAvailablePieces("Hari", 6)
    assert pieces == ["a","d"], "Test6 Failed Hari"
```

In the test7, we check if the specific move is done correctly and if the pieces of each player are located at the right position after playing the move.

```
def test7():
    played = gs.play("Filip", forTestONLY = 'a', dieforTestONLY = 6)
    assert played == (True ,['a',1]), "Test7 Failed Filip"
    played = gs.play("Dora", forTestONLY = 'exit', dieforTestONLY = 3)
    assert played == (False, None), "Test7 Failed Dora"
    played = gs.play("Ana", forTestONLY = 'exit', dieforTestONLY = 2)
    assert played == 'exit', "Test7 Failed Ana"
    played = gs.play("Hari", forTestONLY = 'd', dieforTestONLY = 5)
    assert played == (False ,['d',43]), "Test7 Failed Hari"
    played = gs.play("Filip", forTestONLY = 'a', dieforTestONLY = 6)
    assert played == (True ,['a',7]), "Test7 Failed Filip2"
    played = gs.play("Dora", forTestONLY = 'exit', dieforTestONLY = 6)
    assert played == 'exit', "Test7 Failed Dora2"
    played = gs.play("Hari", forTestONLY = 'd', dieforTestONLY = 1)
    assert played == (False, ['d', 44]), "Test7 Failed Hari2"
    played = gs.play("Hari", forTestONLY = 'b', dieforTestONLY = 1)
    assert played == (False ,['b',42]), "Test7 Failed Hari3"
```

In the test8, we check if a player's pieces are killed after another player's piece steps on them. In the first situation, Filip moves his piece "c" for 6 steps. After that, his pieces "c" and "d" are both located on the relative position 19. We can see that they both survive because they belong to the same player so they can share a place. In the next situation, Dora moves her piece "a" for 4 pieces, placing it on her relative position 36, which corresponds to Filip's relative position 6, where his piece "b" is located. Thus, Filip's "b" is killed by Dora's "a" and it is located at the position 0 (outside the game). The next example shows a similar situation, but this time Ana's piece "a" kills Hari's two pieces "a" and "d" that were located on the same spot. The last situation is like the first one, but it happens inside the house.

```
def test8():
    dic2 = { # relative positions for each player
    "Filip" : {"a":0,"b":6,"c":13,"d":19},
    "Dora" : {"a":32,"b":0,"c":0,"d":0},
    "Ana" : {"a":10,"b":22,"c":41,"d":43},
    "Hari" : {"a":1,"b":41,"c":42,"d":1},
    }
    gs.ps = dic2
    # Filip's "c" cannot kill his "d"
    gs.play("Filip", forTestONLY = 'c', dieforTestONLY = 6)
    assert gs.ps["Filip"]["c"] == 19, "Test8 Failed Filip"
    assert gs.ps["Filip"]["d"] == 19, "Test8 Failed Filip2"
    # Dora's "a" kills Filip's "b"
    gs.play("Dora", forTestONLY = 'a', dieforTestONLY = 4)
    assert gs.ps["Dora"]["a"] == 36, "Test8 Failed Dora"
    assert gs.ps["Filip"]["b"] == 0, "Test8 Failed Filip3"
    # Ana's "a" kills Hari's "a" and "d"
    gs.play("Ana", forTestONLY = 'a', dieforTestONLY = 1)
    assert gs.ps["Ana"]["a"] == 11, "Test8 Failed Ana"
    assert gs.ps["Hari"]["a"] == 0, "Test8 Failed Hari"
    assert gs.ps["Hari"]["d"] == 0, "Test8 Failed Hari2"
    # Hari's "b" cannot kill his "c"
    gs.play("Hari", forTestONLY = 'b', dieforTestONLY = 1)
    assert gs.ps["Hari"]["b"] == 42, "Test8 Failed Hari3"
    assert gs.ps["Hari"]["c"] == 42, "Test8 Failed Hari4"
```

In the first example of the test9, we load new moves and check if the pieces have adjusted their positions correctly. In the next example, we set all Dora's pieces inside the positions of her house (the winning position) and check if the pieces have adjusted their positions correctly.

```
def test9():
    # Dora's rel pos 36 is Filip's rel pos 6, so Filip's "a" kills Dora's "a"
    move = ["a", 6]
    gs.loadMove("Filip", move)
    move = ["b", 0]
    gs.loadMove("Filip", move)
    move = ["c", 0]
    gs.loadMove("Filip", move)
    move = ["d", 0]
    gs.loadMove("Filip", move)
    assert gs.ps["Dora"]["a"] == 0, "Test9 Failed Dora"
    assert gs.ps["Filip"]["a"] == 6, "Test9 Failed Filip"
    assert gs.ps["Filip"]["b"] == 0, "Test9 Failed Filip2"
    assert gs.ps["Filip"]["c"] == 0, "Test9 Failed Filip3"
    assert gs.ps["Filip"]["d"] == 0, "Test9 Failed Filip4"
    # turn Dora into a winner
    move = ["a", 41]
    gs.loadMove("Dora", move)
    move = ["b", 43]
    gs.loadMove("Dora", move)
    move = ["c", 44]
    gs.loadMove("Dora", move)
    move = ["d", 42]
    gs.loadMove("Dora", move)
    assert gs.ps["Dora"]["a"] == 41, "Test9 Failed Dora2"
    assert gs.ps["Dora"]["b"] == 43, "Test9 Failed Dora3"
    assert gs.ps["Dora"]["c"] == 44, "Test9 Failed Dora4"
    assert gs.ps["Dora"]["d"] == 42, "Test9 Failed Dora5"
```

In the test10, we make sure that the method *getWinner* indeed recognises Dora as the winner.

```
def test10():
    winner = gs.getWinner()
    assert winner == "Dora", "Test10 Failed Dora"
```

In the next example, we demonstrate the application's behaviour with 3 players in the game: Dora, Filip, and Hari.

Dora sends the invite.

```
enter name: Dora
Get Address  20000
[CODE] ==> [HELLO] Sending Hello

[CODE] ==> [HELLO] Change in online addresses, {30000}

[CODE] ==> [HELLO] Change in online addresses, {30000, 40000}

invite
[CODE] ==> [INVITE] Send Invitation:
```

Filip accepts the invite, and after him Hari does the same.

```
enter name: Filip
Get Address  30000
[CODE] ==> [HELLO] Sending Hello

[CODE] ==> [HELLO_RPL] Change in online addresses, {20000}

[CODE] ==> [HELLO] Change in online addresses, {20000, 40000}

[CODE] ==> [INVITE] Recieve invite from Dora

acc_invite
```

```
enter name: Hari
Get Address  40000
[CODE] ==> [HELLO] Sending Hello

[CODE] ==> [HELLO_RPL] Change in online addresses, {20000}

[CODE] ==> [HELLO_RPL] Change in online addresses, {20000, 30000}

[CODE] ==> [INVITE] Recieve invite from Dora

acc_invite
```

Players are added to the list of players according to the order of accepting the invite: after Dora who is the initiator, next in the list is Filip, and after him Hari. Dora starts the game, so she becomes the leader of the game.

```
[CODE] ==> [ACC_INVITE] Players: ['Dora', 'Filip']

[CODE] ==> [ACC_INVITE] Players: ['Dora', 'Filip', 'Hari']

start
[CODE] ==> [START] Send Players: ['Dora', 'Filip', 'Hari']
[GAME INTERFACE] ==> Your turn to play, write play to play move
```

Dora plays, she gets the number 6 on the dice and the pieces available to move are shown on the screen.

```
[GAME INTERFACE] ==> Your turn to play, write play to play move

play
[GAME INTERFACE] ==> Your number on die is: 6
[GAME INTERFACE] ==> You can move on of following piceses: ['a', 'b', 'c', 'd']
```

All four pieces are initially outside the game, so she can choose any of the, for example the piece “c”.

```
c
[CODE] ==> [PLAYED MOVE] Sending move: Dora<>True<>['c', 1]
[CODE] ==> [PLAYED MOVE] Recive move: Dora<>True<>['c', 1]
[GAME INTERFACE] ==> {'Dora': {'a': 0, 'b': 0, 'c': 1, 'd': 0}, 'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Hari': {'a': 0, 'b': 0, 'c': 0, 'd': 0}}
[CODE] ==> [ON_MOVE] Sending on move
[GAME INTERFACE] ==> Your turn to play, write play to play move
```

Now her piece “c” is located at the relative position 1, while all the other pieces are still outside the game (location 0). Since she got a 6 on the dice, she continues to play.

```
[GAME INTERFACE] ==> Your turn to play, write play to play move


[GAME INTERFACE] ==> Your number on die is: 6
[GAME INTERFACE] ==> You can move on of following piceses: ['a', 'b', 'c', 'd']

a
[CODE] ==> [PLAYED MOVE] Sending move: Dora<>True<>['a', 1]
[CODE] ==> [PLAYED MOVE] Recive move: Dora<>True<>['a', 1]
[GAME INTERFACE] ==> {'Dora': {'a': 1, 'b': 0, 'c': 1, 'd': 0}, 'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Hari': {'a': 0, 'b': 0, 'c': 0, 'd': 0}}
[CODE] ==> [ON_MOVE] Sending on move
[GAME INTERFACE] ==> Your turn to play, write play to play move


play
[GAME INTERFACE] ==> Your number on die is: 1
[GAME INTERFACE] ==> You can move on of following piceses: ['a', 'c']

c
[CODE] ==> [PLAYED MOVE] Sending move: Dora<>False<>['c', 2]
[CODE] ==> [PLAYED MOVE] Recive move: Dora<>False<>['c', 2]
[GAME INTERFACE] ==> {'Dora': {'a': 1, 'b': 0, 'c': 2, 'd': 0}, 'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Hari': {'a': 0, 'b': 0, 'c': 0, 'd': 0}}
[CODE] ==> [ON_MOVE] Sending on move
```

Next, it is Filip’s turn, because he is the next player in the list of players. He gets number 4 on the dice, but all his pieces are still outside the game, so he cannot move them. The next player on the move is Hari.

```
[CODE] ==> [ON_MOVE] Players on move: Filip
[GAME INTERFACE] ==> Your turn to play, write play to play move

play
[GAME INTERFACE] ==> Your number on die is: 4
[GAME INTERFACE] ==> You can not move any of your pieces
[CODE] ==> [PLAYED MOVE] Sending move: Filip<>False<>None
[CODE] ==> [PLAYED MOVE] Recive move: Filip<>False<>None
[GAME INTERFACE] ==> {'Dora': {'a': 0, 'b': 0, 'c': 2, 'd': 0}, 'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Hari': {'a': 0, 'b': 0, 'c': 0, 'd': 0}}


[CODE] ==> [ON_MOVE] Players on move: Hari
```

Hari gets a 6 and chooses to take the piece “a” into the game. Since he got a 6, he continues to play.

```
play
[GAME INTERFACE] ==> Your number on die is: 6
[GAME INTERFACE] ==> You can move on of following piceses: ['a', 'b', 'c', 'd']

a
```

In the next move he gets a 6 again and chooses to take the piece “b” into the game. Now he has two pieces in the game, “a” and “b”. Since he got a 6, he continues to play.

In the next move he gets a 5 and he moves the piece “a” for 5 steps forward, so the next player on the move is Dora. Dora plays her move.

```
play
[GAME INTERFACE] ==> Your number on die is: 5
[GAME INTERFACE] ==> You can move on of following piceses: ['a', 'b']

a
[CODE] ==> [PLAYED MOVE] Sending move: Hari<>False<>['a', 6]
[CODE] ==> [PLAYED MOVE] Recive move: Hari<>False<>['a', 6]
[GAME INTERFACE] ==> {'Dora': {'a': 0, 'b': 0, 'c': 2, 'd': 0}, 'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Hari': {'a': 6, 'b': 1, 'c': 0, 'd': 0}}


[CODE] ==> [ON_MOVE] Players on move: Dora
```

The game continues.

There are a few interesting scenarios that might happen during the game. We will describe those situations and demonstrate how our application behaves when it encounters them.

Firstly, we demonstrate playing the forbidden moves through the following example.

It is Dora’s turn to play, but Filip tries to play when it is not his turn. In that situation, he gets an appropriate message which tells him that he cannot play.

```
[CODE] ==> [ON_MOVE] Players on move: Dora

play
[GAME INTERFACE] ==> It is not your move.
```

Secondly, we will demonstrate various situations of exiting the game.

In any moment, any player can decide to exit the game. If a non-leader player exits the game, he is just removed from the list of players and the game continues normally for the rest of the players.

For example, if Filip exits the game while he is on the move, he will be removed from the list of players and he will simply be skipped, so the next player on the move is Hari.

```
[CODE] ==> [ON_MOVE] Players on move: Filip
[GAME INTERFACE] ==> Your turn to play, write play to play move

exit
[CODE] ==> [ON_MOVE] Players on move: Hari
```

A player can exit the game even if they are not on the move. For instance, after Hari gets a 5 on the dice and decides to move his piece “b”, it is Dora’s turn again. Then, Hari decides to exit the game while it is Dora’s turn.

```
[CODE] ==> [ON_MOVE] Players on move: Hari
[GAME INTERFACE] ==> Your turn to play, write play to play move

play
[GAME INTERFACE] ==> Your number on die is: 5
[GAME INTERFACE] ==> You can move on of following piceses: ['a', 'b']

b
[CODE] ==> [PLAYED MOVE] Sending move: Hari<>False<>['b', 6]
[CODE] ==> [PLAYED MOVE] Recive move: Hari<>False<>['b', 6]
[GAME INTERFACE] ==> {'Dora': {'a': 0, 'b': 0, 'c': 3, 'd': 0}, 'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Hari': {'a': 6, 'b': 6, 'c': 0, 'd': 0}}


[CODE] ==> [ON_MOVE] Players on move: Dora

exit
```

After Hari exits the game, he is also removed from the list of players, so Dora is left as the only player in the game. She is still the leader, just like at the beginning. Since she is the only player left, she continues to play by herself, rolling the dice and moving her pieces. The next picture shows a few of her moves.

```
play
[GAME INTERFACE] ==> Your number on die is: 5
[GAME INTERFACE] ==> You can move on of following piceses: ['c']

[GAME INTERFACE] ==> {'Dora': {'a': 0, 'b': 0, 'c': 16, 'd': 0}, 'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Hari': {'a': 6, 'b': 6, 'c': 0, 'd': 0}}
[CODE] ==> [ON_MOVE] Sending on move
[GAME INTERFACE] ==> Your turn to play, write play to play move


play
[GAME INTERFACE] ==> Your number on die is: 4
[GAME INTERFACE] ==> You can move on of following piceses: ['c']

c
[CODE] ==> [PLAYED MOVE] Sending move: Dora<>False<>['c', 20]
[CODE] ==> [PLAYED MOVE] Recive move: Dora<>False<>['c', 20]
[GAME INTERFACE] ==> {'Dora': {'a': 0, 'b': 0, 'c': 20, 'd': 0}, 'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Hari': {'a': 6, 'b': 6, 'c': 0, 'd': 0}}
[CODE] ==> [ON_MOVE] Sending on move
[GAME INTERFACE] ==> Your turn to play, write play to play move
```

After some time, Hari returns to the game.

```
enter name: Hari
Get Address  30000
[CODE] ==> [HELLO] Sending Hello

[CODE] ==> [HELLO_RPL] Change in online addresses, {20000}

[CODE] ==> [RETURN] I am back to the game
```

Dora also gets notified about his return.

```
[GAME INTERFACE] ==> Your turn to play, write play to play move


[CODE] ==> [HELLO] Change in online addresses, {30000}
[CODE] ==> [RETURN] Hari is back to the game

[CODE] ==> [RETURN] Inform all players about the returning player
```

After she plays her move, it is Hari's time to play. We can see that the application remembers the positions of his pieces from before he exited the game, so the information is easily restored. After he plays his move, it is Dora's turn again.

```
[CODE] ==> [ON_MOVE] Players on move: Hari
[GAME INTERFACE] ==> Your turn to play, write play to play move

play
[GAME INTERFACE] ==> Your number on die is: 5
[GAME INTERFACE] ==> You can move on of following piceses: ['a', 'b']

b
[CODE] ==> [PLAYED MOVE] Sending move: Hari<>False<>['b', 11]
[CODE] ==> [PLAYED MOVE] Recive move: Hari<>False<>['b', 11]
[GAME INTERFACE] ==> {'Dora': {'a': 0, 'b': 0, 'c': 24, 'd': 0}, 'Hari': {'a': 6, 'b': 11, 'c': 0, 'd': 0}, 'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}}


[CODE] ==> [ON_MOVE] Players on move: Dora
```

But, in this moment, Filip also returns to the game.

```
enter name: Filip
Get Address  40000
[CODE] ==> [HELLO] Sending Hello

[CODE] ==> [HELLO_RPL] Change in online addresses, {20000}

[CODE] ==> [HELLO_RPL] Change in online addresses, {20000, 30000}

[CODE] ==> [RETURN] I am back to the game
```

Both Dora and Hari get notified about his return.

Dora's view

```
[GAME INTERFACE] ==> Your turn to play, write play to play move

[CODE] ==> [HELLO] Change in online addresses, {30000, 40000}
[CODE] ==> [RETURN] Filip is back to the game

[CODE] ==> [RETURN] Inform all players about the returning player
```

Hari's view

```
[CODE] ==> [ON_MOVE] Players on move: Dora

[CODE] ==> [HELLO] Change in online addresses, {20000, 40000}

[CODE] ==> [RETURN] Filip is back to the game
```

Dora plays her move.

```
[GAME INTERFACE] ==> Your number on die is: 4
[GAME INTERFACE] ==> You can move on of following piceses: ['c']

c
[CODE] ==> [PLAYED MOVE] Sending move: Dora<>False<>['c', 28]
[CODE] ==> [PLAYED MOVE] Recive move: Dora<>False<>['c', 28]
[GAME INTERFACE] ==> {'Dora': {'a': 0, 'b': 0, 'c': 28, 'd': 0}, 'Hari': {'a': 6, 'b': 11, 'c': 0, 'd': 0}, 'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}}
[CODE] ==> [ON_MOVE] Sending on move
[CODE] ==> [UPDATE_STATE] Send Upadated state to adress: {40000}
```

After Dora, the next player will be Hari, because he returned to the game before Filip.

```
[CODE] ==> [ON_MOVE] Players on move: Hari
[GAME INTERFACE] ==> Your turn to play, write play to play move

play
[GAME INTERFACE] ==> Your number on die is: 5
[GAME INTERFACE] ==> You can move on of following piceses: ['a', 'b']

a
[CODE] ==> [PLAYED MOVE] Sending move: Hari<>False<>['a', 11]
[CODE] ==> [PLAYED MOVE] Recive move: Hari<>False<>['a', 11]
[GAME INTERFACE] ==> {'Dora': {'a': 0, 'b': 0, 'c': 28, 'd': 0}, 'Hari': {'a': 11, 'b': 11, 'c': 0, 'd': 0}, 'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}}


[CODE] ==> [ON_MOVE] Players on move: Filip
```

After Hari, it is now Filip's turn to play his move.

```
[GAME INTERFACE] ==> Your turn to play, write play to play move

play
[GAME INTERFACE] ==> Your number on die is: 3
[GAME INTERFACE] ==> You can not move any of your pieces
[CODE] ==> [PLAYED MOVE] Sending move: Filip<>False<>None
[CODE] ==> [PLAYED MOVE] Recive move: Filip<>False<>None
[GAME INTERFACE] ==> {'Dora': {'a': 0, 'b': 0, 'c': 28, 'd': 0}, 'Hari': {'a': 11, 'b': 11, 'c': 0, 'd': 0}, 'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}}


[CODE] ==> [ON_MOVE] Players on move: Dora
```

We can see that if some players exit the game and then return in a different order than before, the new order of playing is determined according to the order of return.

The next situation we will demonstrate is leader's exit. If the current leader exits the game, the new leader is chosen as the next player from the list of players, which is now Hari.

```
[GAME INTERFACE] ==> Your turn to play, write play to play move

exit
[CODE] ==> [LEADER] Changed Leader: Hari

[CODE] ==> [ON_MOVE] Players on move: Hari
```

Eventually, the ex-leader can return to the game, but this does not change the current leader.

```
enter name: Dora
Get Address  20000
[CODE] ==> [HELLO] Sending Hello

[CODE] ==> [HELLO_RPL] Change in online addresses, {30000}

[CODE] ==> [HELLO_RPL] Change in online addresses, {30000, 40000}

[CODE] ==> [RETURN] I am back to the game
```

The current leader is still Hari, and he informs all players about the return.

```
[CODE] ==> [HELLO] Change in online addresses, {20000, 40000}
[CODE] ==> [RETURN] Dora is back to the game

[CODE] ==> [RETURN] Inform all players about the returning player
```

The next situation we will demonstrate is when one player's piece steps on another player's place and "kills" them (casts them out of the game).

We can see that Dora's piece "c" is located on her relative position 29, and Hari's piece "b" is located on his relative position 14.

```
[GAME INTERFACE] ==> {'Dora': {'a': 0, 'b': 0, 'c': 29, 'd': 0}, 'Hari': {'a': 11, 'b': 14, 'c': 0, 'd': 0}, 'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}}
[CODE] ==> [ON_MOVE] Sending on move
```

After Hari gets a 5 on the dice, he moves his piece "b" 5 steps forward and comes to his relative position 19, which equals to Dora's relative position 29. Since Hari's piece "b" came to the same position where Dora's piece "c" was standing, Dora's "c" is cats out of the game back to the position 0.

```
play
[GAME INTERFACE] ==> Your number on die is: 5
[GAME INTERFACE] ==> You can move on of following piceses: ['a', 'b']

b
[CODE] ==> [PLAYED MOVE] Sending move: Hari<>False<>['b', 19]
[CODE] ==> [PLAYED MOVE] Recive move: Hari<>False<>['b', 19]
[GAME INTERFACE] ==> {'Dora': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Hari': {'a': 11, 'b': 19, 'c': 0, 'd': 0}, 'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}}
```

If a player's piece steps on another one of their own pieces, they do not kill each other, but they keep standing on the same position.

Finally, we will demonstrate the situation in which one of the players eventually wins the game.

After some time, Filip manages to put all his pieces into the house and thus wins the game. All the players get the message that Filip won the game.

```
[GAME INTERFACE] ==> {'Filip': {'a': 44, 'b': 42, 'c': 43, 'd': 39}, 'Dora': {'a': 44, 'b': 43, 'c': 41, 'd': 38}, 'Hari': {'a': 44, 'b': 40, 'c': 40, 'd': 40}}

[CODE] ==> [ON_MOVE] Players on move: Filip
[GAME INTERFACE] ==> Your turn to play, write play to play move

play
[GAME INTERFACE] ==> Your number on die is: 2
[GAME INTERFACE] ==> You can move on of following piceses: ['d']

d
[CODE] ==> [PLAYED MOVE] Sending move: Filip<>False<>['d', 41]
[CODE] ==> [PLAYED MOVE] Recive move: Filip<>False<>['d', 41]
[GAME INTERFACE] ==> {'Filip': {'a': 44, 'b': 42, 'c': 43, 'd': 41}, 'Dora': {'a': 44, 'b': 43, 'c': 41, 'd': 38}, 'Hari': {'a': 44, 'b': 40, 'c': 40, 'd': 40}}


Player: Filip is the winner!!
```

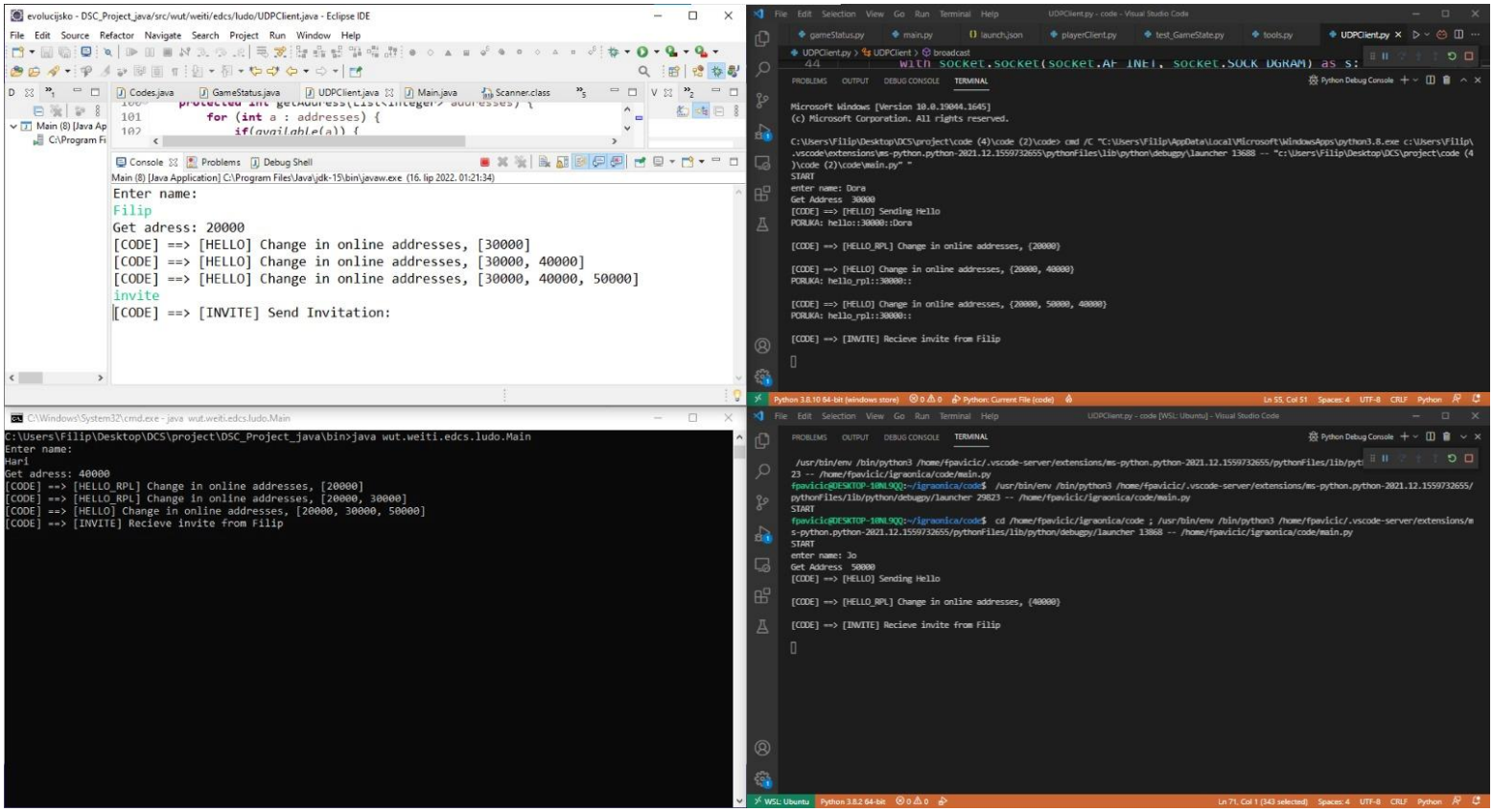
After one of the player wins, the game is over.

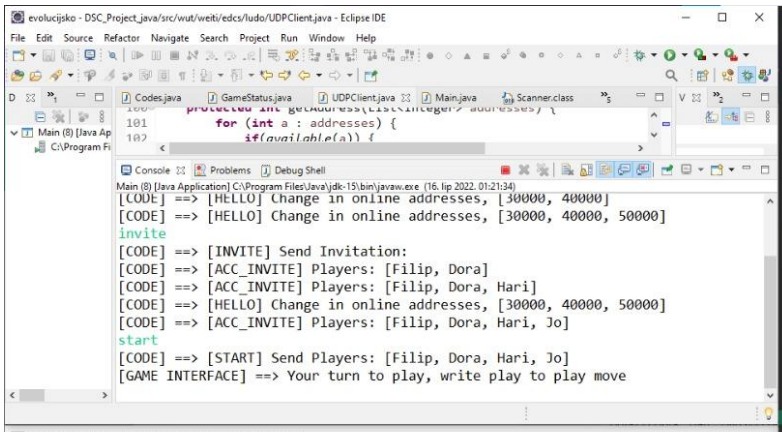
7. Working in different programming languages and operational systems

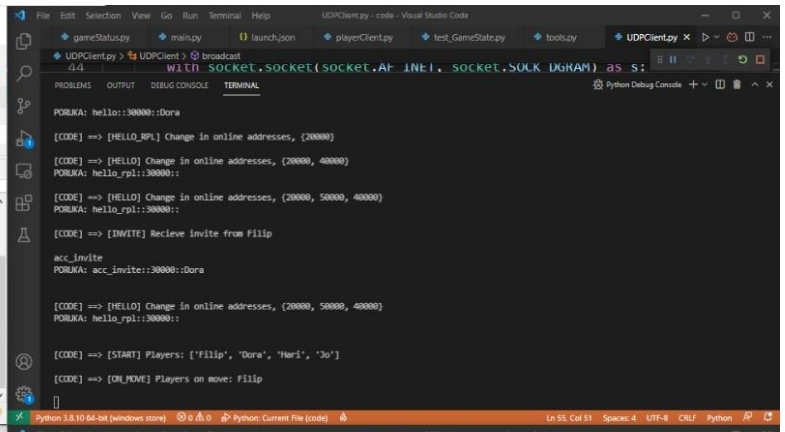
We implemented our application in two programming languages: Java and Python. To show that our distributed system works in different operational systems, we created 4 player clients in:

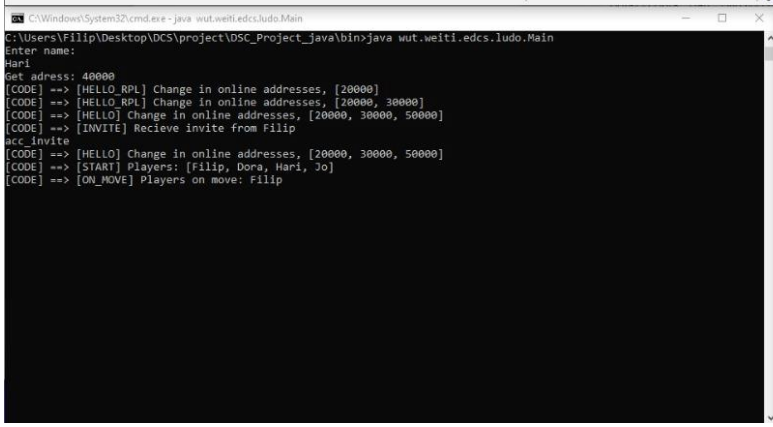
- Windows 10 + Java in Eclipse IDE
- Linux (Ubuntu) + Python in Terminal
- Windows 10 + Java Command Line Prompt
- Windows 10 + Python Command Line Prompt

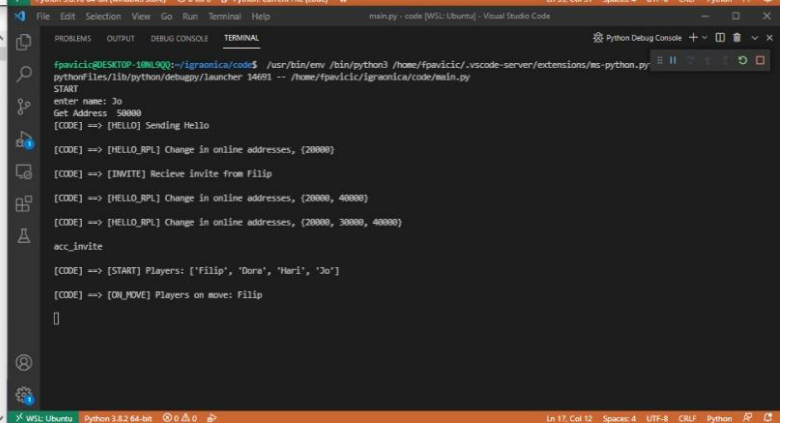
Here are some examples of running the application with the clients described above.

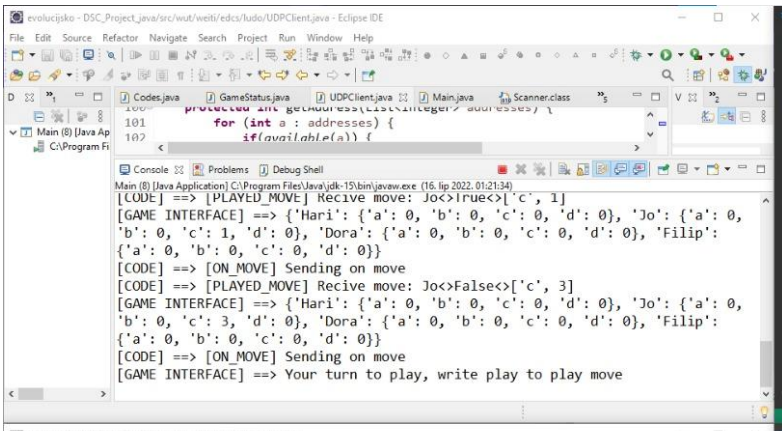


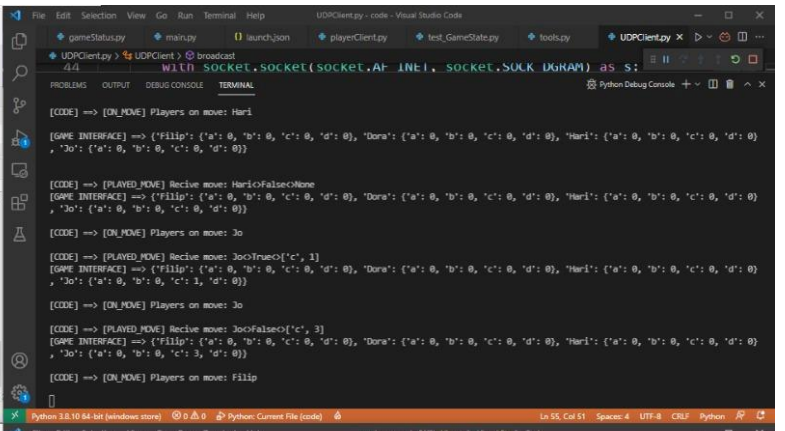

evolucijsko - DSC_Project_java/src/wut/weiti/edcs/ludo/UDPClient.java - Eclipse IDE
Console
Problems
Debug Shell
Main (8) [Java Application] C:\Program Files\Java\jdk-15\bin\javaw.exe (16. lip 2022. 01:21:34)
[CODE] ==> [HELLO] Change in online addresses, [30000, 40000]
[CODE] ==> [HELLO] Change in online addresses, [30000, 40000, 50000]
invite
[CODE] ==> [INVITE] Send Invitation:
[CODE] ==> [ACC_INVITE] Players: [Filip, Dora]
[CODE] ==> [ACC_INVITE] Players: [Filip, Dora, Hari]
[CODE] ==> [HELLO] Change in online addresses, [30000, 40000, 50000]
[CODE] ==> [ACC_INVITE] Players: [Filip, Dora, Hari, Jo]
start
[CODE] ==> [START] Send Players: [Filip, Dora, Hari, Jo]
[GAME INTERFACE] ==> Your turn to play, write play to play move

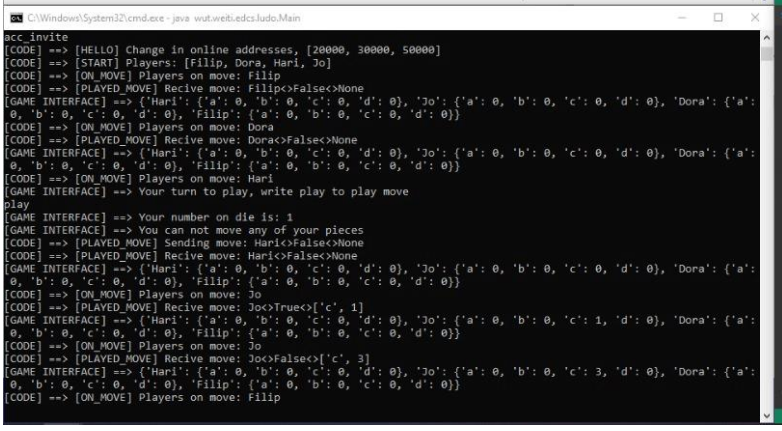

UDPClient.py - code - Visual Studio Code
PORUKA: hello::30000::Dora
[CODE] ==> [HELLO_RPL] Change in online addresses, {20000}
[CODE] ==> [HELLO] Change in online addresses, {20000, 40000}
PORUKA: hello_rpl::30000::
[CODE] ==> [HELLO] Change in online addresses, {20000, 50000, 40000}
PORUKA: hello_rpl::30000::
[CODE] ==> [INVITE] Recieve invite from Filip
acc_invite
PORUKA: acc_invite::30000::Dora
[CODE] ==> [HELLO] Change in online addresses, {20000, 50000, 40000}
PORUKA: hello_rpl::30000::
[CODE] ==> [START] Players: ['Filip', 'Dora', 'Hari', 'Jo']
[CODE] ==> [ON_MOVE] Players on move: Filip

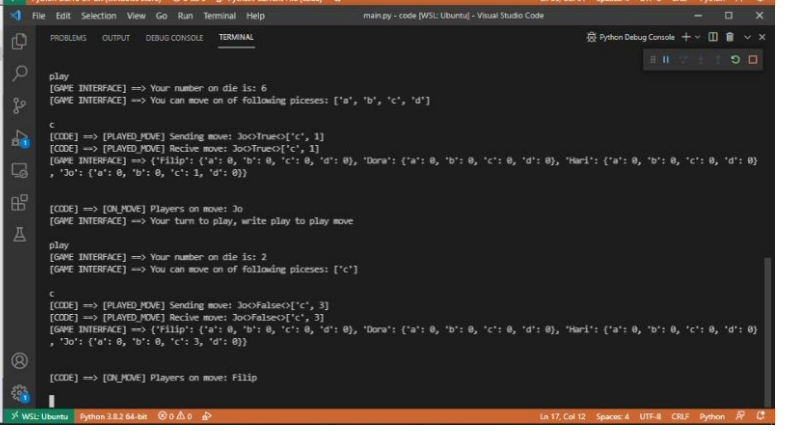

C:\Windows\System32\cmd.exe - java wut.weiti.edcs.ludo.Main
C:\Users\Filip\Desktop\DCS\project\DSC_Project_java\bin>java wut.weiti.edcs.ludo.Main
Enter name:
Hari
Get adress: 40000
[CODE] ==> [HELLO_RPL] Change in online addresses, [20000]
[CODE] ==> [HELLO_RPL] Change in online addresses, [20000, 30000]
[CODE] ==> [HELLO] Change in online addresses, [20000, 30000, 50000]
[CODE] ==> [INVITE] Recieve invite from Filip
acc_invite
[CODE] ==> [HELLO] Change in online addresses, [20000, 30000, 50000]
[CODE] ==> [START] Players: [Filip, Dora, Hari, Jo]
[CODE] ==> [ON_MOVE] Players on move: Filip

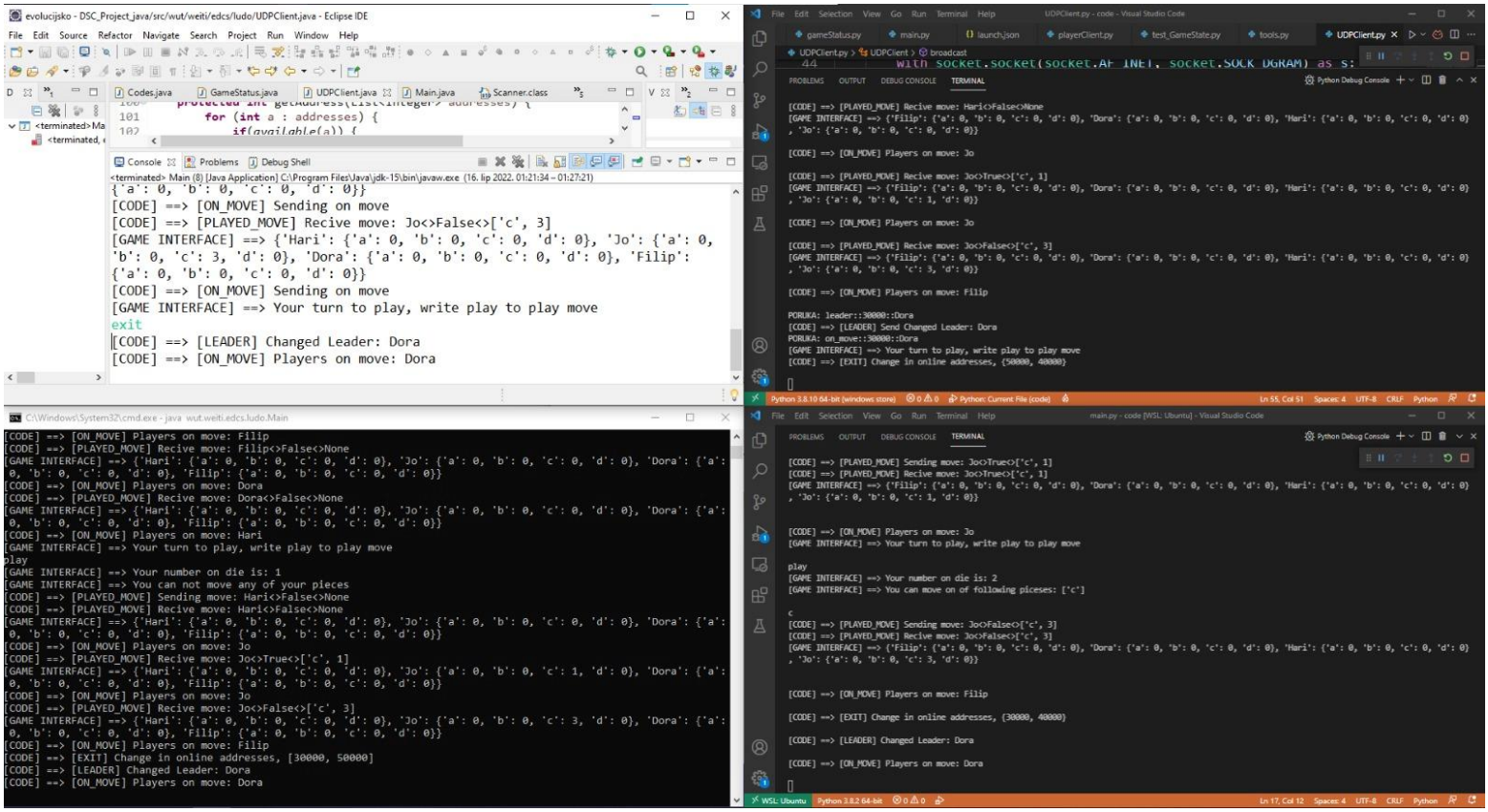

main.py - code [WSL: Ubuntu] - Visual Studio Code
fpavicic@DESKTOP-10NL9QQ:~/igraonica/code$ /usr/bin/env /bin/python3 /home/fpavicic/.vscode-server/extensions/ms-python.py
START
enter name: Jo
Get Address 50000
[CODE] ==> [HELLO] Sending Hello
[CODE] ==> [HELLO_RPL] Change in online addresses, {20000}
[CODE] ==> [INVITE] Recieve invite from Filip
[CODE] ==> [HELLO_RPL] Change in online addresses, {20000, 40000}
[CODE] ==> [HELLO_RPL] Change in online addresses, {20000, 30000, 40000}
acc_invite
[CODE] ==> [START] Players: ['Filip', 'Dora', 'Hari', 'Jo']
[CODE] ==> [ON_MOVE] Players on move: Filip


Console
Problems
Debug Shell
Main (8) [Java Application] C:\Program Files\Java\jdk-15\bin\javaw.exe (16. lip 2022. 01:21:34)
[CODE] ==> [PLAYED_MOVE] Recive move: Jo<>True<>['c', 1]
[GAME INTERFACE] ==> {'Hari': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Jo': {'a': 0, 'b': 0, 'c': 1, 'd': 0}, 'Dora': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}}
[CODE] ==> [ON_MOVE] Sending on move
[CODE] ==> [PLAYED_MOVE] Recive move: Jo<>False<>['c', 3]
[GAME INTERFACE] ==> {'Hari': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Jo': {'a': 0, 'b': 0, 'c': 3, 'd': 0}, 'Dora': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}}
[CODE] ==> [ON_MOVE] Sending on move
[GAME INTERFACE] ==> Your turn to play, write play to play move


[CODE] ==> [ON_MOVE] Players on move: Hari
[GAME INTERFACE] ==> {'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Dora': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Hari': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Jo': {'a': 0, 'b': 0, 'c': 0, 'd': 0}}
[CODE] ==> [PLAYED_MOVE] Recive move: Hari<>False<>None
[GAME INTERFACE] ==> {'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Dora': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Hari': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Jo': {'a': 0, 'b': 0, 'c': 0, 'd': 0}}
[CODE] ==> [ON_MOVE] Players on move: Jo
[CODE] ==> [PLAYED_MOVE] Recive move: Jo<>True<>['c', 1]
[GAME INTERFACE] ==> {'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Dora': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Hari': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Jo': {'a': 0, 'b': 0, 'c': 1, 'd': 0}}
[CODE] ==> [ON_MOVE] Players on move: Jo
[CODE] ==> [PLAYED_MOVE] Recive move: Jo<>False<>['c', 3]
[GAME INTERFACE] ==> {'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Dora': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Hari': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Jo': {'a': 0, 'b': 0, 'c': 3, 'd': 0}}
[CODE] ==> [ON_MOVE] Players on move: Filip


C:\Windows\System32\cmd.exe - java wut.weiti.edcs.ludo.Main
acc_invite
[CODE] ==> [HELLO] Change in online addresses, [20000, 30000, 50000]
[CODE] ==> [START] Players: [Filip, Dora, Hari, Jo]
[CODE] ==> [ON_MOVE] Players on move: Filip
[CODE] ==> [PLAYED_MOVE] Recive move: Filip<>False<>None
[GAME INTERFACE] ==> {'Hari': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Jo': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Dora': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}}
[CODE] ==> [ON_MOVE] Players on move: Dora
[CODE] ==> [PLAYED_MOVE] Recive move: Dora<>False<>None
[GAME INTERFACE] ==> {'Hari': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Jo': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Dora': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}}
[CODE] ==> [ON_MOVE] Players on move: Hari
[GAME INTERFACE] ==> Your turn to play, write play to play move
play
[GAME INTERFACE] ==> Your number on die is: 1
[GAME INTERFACE] ==> You can not move any of your pieces
[CODE] ==> [PLAYED_MOVE] Sending move: Hari<>False<>None
[CODE] ==> [PLAYED_MOVE] Recive move: Hari<>False<>None
[GAME INTERFACE] ==> {'Hari': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Jo': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Dora': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}}
[CODE] ==> [ON_MOVE] Players on move: Jo
[CODE] ==> [PLAYED_MOVE] Recive move: Jo<>True<>['c', 1]
[GAME INTERFACE] ==> {'Hari': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Jo': {'a': 0, 'b': 0, 'c': 1, 'd': 0}, 'Dora': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}}
[CODE] ==> [ON_MOVE] Players on move: Jo
[CODE] ==> [PLAYED_MOVE] Recive move: Jo<>False<>['c', 3]
[GAME INTERFACE] ==> {'Hari': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Jo': {'a': 0, 'b': 0, 'c': 3, 'd': 0}, 'Dora': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}}
[CODE] ==> [ON_MOVE] Players on move: Filip


main.py - code [WSL: Ubuntu] - Visual Studio Code
play
[GAME INTERFACE] ==> Your number on die is: 6
[GAME INTERFACE] ==> You can move on of following piceses: ['a', 'b', 'c', 'd']
c
[CODE] ==> [PLAYED_MOVE] Sending move: Jo<>True<>['c', 1]
[CODE] ==> [PLAYED_MOVE] Recive move: Jo<>True<>['c', 1]
[GAME INTERFACE] ==> {'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Dora': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Hari': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Jo': {'a': 0, 'b': 0, 'c': 1, 'd': 0}}
[CODE] ==> [ON_MOVE] Players on move: Jo
[GAME INTERFACE] ==> Your turn to play, write play to play move
play
[GAME INTERFACE] ==> Your number on die is: 2
[GAME INTERFACE] ==> You can move on of following piceses: ['c']
c
[CODE] ==> [PLAYED_MOVE] Sending move: Jo<>False<>['c', 3]
[CODE] ==> [PLAYED_MOVE] Recive move: Jo<>False<>['c', 3]
[GAME INTERFACE] ==> {'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Dora': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Hari': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Jo': {'a': 0, 'b': 0, 'c': 3, 'd': 0}}
[CODE] ==> [ON_MOVE] Players on move: Filip


evolucijsko - DSC_Project_java/src/wut/weiti/edcs/ludo/UDPClient.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Codes.java
GameStatus.java
UDPClient.java
Main.java
Scanner.class
101 for (int a : addresses) {
102 if(available(a)) {
Console
Problems
Debug Shell
<terminated> Main (8) [Java Application] C:\Program Files\Java\jdk-15\bin\javaw.exe (16. lip 2022. 01:21:34 – 01:27:21)
{'a': 0, 'b': 0, 'c': 0, 'd': 0}}
[CODE] ==> [ON_MOVE] Sending on move
[CODE] ==> [PLAYED_MOVE] Recive move: Jo<>False<>['c', 3]
[GAME INTERFACE] ==> {'Hari': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Jo': {'a': 0, 'b': 0, 'c': 3, 'd': 0}, 'Dora': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}}
[CODE] ==> [ON_MOVE] Sending on move
[GAME INTERFACE] ==> Your turn to play, write play to play move
exit
[CODE] ==> [LEADER] Changed Leader: Dora
[CODE] ==> [ON_MOVE] Players on move: Dora
UDPClient.py - code - Visual Studio Code
gameStatus.py
main.py
launch.json
playerClient.py
test_GameState.py
tools.py
UDPClient.py
UDPClient.py > UDPClient > broadcast
44 with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Python Debug Console
[CODE] ==> [PLAYED_MOVE] Recive move: Hari<>False<>None
[GAME INTERFACE] ==> {'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Dora': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Hari': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Jo': {'a': 0, 'b': 0, 'c': 0, 'd': 0}}
[CODE] ==> [ON_MOVE] Players on move: Jo
[CODE] ==> [PLAYED_MOVE] Recive move: Jo<>True<>['c', 1]
[GAME INTERFACE] ==> {'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Dora': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Hari': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Jo': {'a': 0, 'b': 0, 'c': 1, 'd': 0}}
[CODE] ==> [ON_MOVE] Players on move: Jo
[CODE] ==> [PLAYED_MOVE] Recive move: Jo<>False<>['c', 3]
[GAME INTERFACE] ==> {'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Dora': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Hari': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Jo': {'a': 0, 'b': 0, 'c': 3, 'd': 0}}
[CODE] ==> [ON_MOVE] Players on move: Filip
PORUKA: leader::30000::Dora
[CODE] ==> [LEADER] Send Changed Leader: Dora
PORUKA: on_move::30000::Dora
[GAME INTERFACE] ==> Your turn to play, write play to play move
[CODE] ==> [EXIT] Change in online addresses, {50000, 40000}
Python 3.8.10 64-bit (windows store)
Python: Current File (code)
Ln 55, Col 51
Spaces: 4
UTF-8
CRLF
Python
C:\Windows\System32\cmd.exe - java wut.weiti.edcs.ludo.Main
[CODE] ==> [ON_MOVE] Players on move: Filip
[CODE] ==> [PLAYED_MOVE] Recive move: Filip<>False<>None
[GAME INTERFACE] ==> {'Hari': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Jo': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Dora': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}}
[CODE] ==> [ON_MOVE] Players on move: Dora
[CODE] ==> [PLAYED_MOVE] Recive move: Dora<>False<>None
[GAME INTERFACE] ==> {'Hari': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Jo': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Dora': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}}
[CODE] ==> [ON_MOVE] Players on move: Hari
[GAME INTERFACE] ==> Your turn to play, write play to play move
play
[GAME INTERFACE] ==> Your number on die is: 1
[GAME INTERFACE] ==> You can not move any of your pieces
[CODE] ==> [PLAYED_MOVE] Sending move: Hari<>False<>None
[CODE] ==> [PLAYED_MOVE] Recive move: Hari<>False<>None
[GAME INTERFACE] ==> {'Hari': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Jo': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Dora': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}}
[CODE] ==> [ON_MOVE] Players on move: Jo
[CODE] ==> [PLAYED_MOVE] Recive move: Jo<>True<>['c', 1]
[GAME INTERFACE] ==> {'Hari': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Jo': {'a': 0, 'b': 0, 'c': 1, 'd': 0}, 'Dora': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}}
[CODE] ==> [ON_MOVE] Players on move: Jo
[CODE] ==> [PLAYED_MOVE] Recive move: Jo<>False<>['c', 3]
[GAME INTERFACE] ==> {'Hari': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Jo': {'a': 0, 'b': 0, 'c': 3, 'd': 0}, 'Dora': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}}
[CODE] ==> [ON_MOVE] Players on move: Filip
[CODE] ==> [EXIT] Change in online addresses, [30000, 50000]
[CODE] ==> [LEADER] Changed Leader: Dora
[CODE] ==> [ON_MOVE] Players on move: Dora
main.py - code [WSL: Ubuntu] - Visual Studio Code
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Python Debug Console
[CODE] ==> [PLAYED_MOVE] Sending move: Jo<>True<>['c', 1]
[CODE] ==> [PLAYED_MOVE] Recive move: Jo<>True<>['c', 1]
[GAME INTERFACE] ==> {'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Dora': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Hari': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Jo': {'a': 0, 'b': 0, 'c': 1, 'd': 0}}
[CODE] ==> [ON_MOVE] Players on move: Jo
[GAME INTERFACE] ==> Your turn to play, write play to play move
play
[GAME INTERFACE] ==> Your number on die is: 2
[GAME INTERFACE] ==> You can move on of following piceses: ['c']
c
[CODE] ==> [PLAYED_MOVE] Sending move: Jo<>False<>['c', 3]
[CODE] ==> [PLAYED_MOVE] Recive move: Jo<>False<>['c', 3]
[GAME INTERFACE] ==> {'Filip': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Dora': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Hari': {'a': 0, 'b': 0, 'c': 0, 'd': 0}, 'Jo': {'a': 0, 'b': 0, 'c': 3, 'd': 0}}
[CODE] ==> [ON_MOVE] Players on move: Filip
[CODE] ==> [EXIT] Change in online addresses, {30000, 40000}
[CODE] ==> [LEADER] Changed Leader: Dora
[CODE] ==> [ON_MOVE] Players on move: Dora
WSL: Ubuntu
Python 3.8.2 64-bit
Ln 17, Col 12
Spaces: 4
UTF-8
CRLF
Python