

# Drop Disc Game

## About the game:

Drop Disc, a variation of the famous Connect Four game, is a two-player connection game in which the players first choose a color and then take turns dropping colored discs from the top into a nine-column, six-row vertically suspended grid. The pieces fall straight down, occupying the next available space within the column.

The objective of the game is to be the first to form a horizontal, vertical, or diagonal line of five of one's own discs.

## Setting up the Game Application:

1. The game follows a client-server architecture, The server side application and the client side application are written in two different repos for easy maintainability.
  - a. Please clone the following repos into your local system.
    - i. Server: <https://github.com/mathew55/drop-disc-server>
    - ii. Client: <https://github.com/mathew55/drop-disc-client>
2. Both client & server is written using python 3.9.5, Hence python 3.9.5 is the ideally preferred version to run the application. However, the code itself does not depend heavily on any specific libraries, Any version from 3.7 should work ideally.

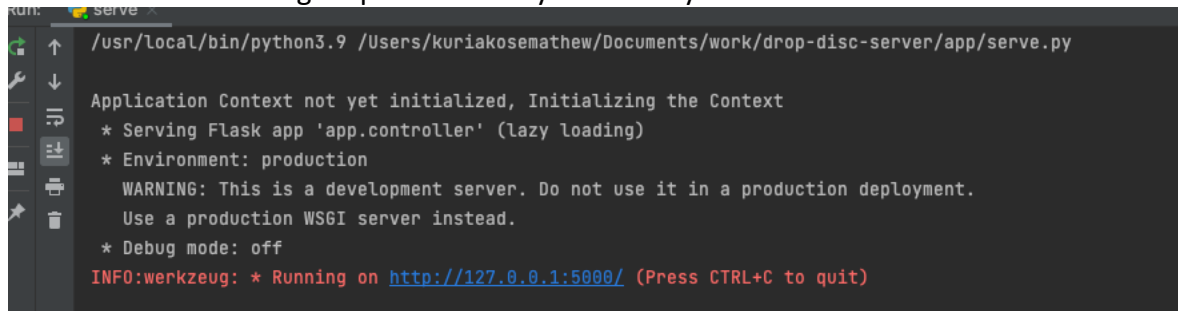
**Note:** 3.9 is still preferred to match the compatibility between test environment and development environment. I had plans to put the logic in a docker, but was not sure if that was worth the effort as most companies have limitations on the docker images which can run on the system.

3. Both client and server uses few external libraries, the external dependencies are captured in a requirements.txt file in the respective repositories. Please follow the steps here to install the requirements:
  - a. Create a new virtualenv for both client and server.
  - b. Activate the virtualenv
  - c. **CD** to the directory where requirements.txt is located
  - d. Run `pip install -r requirement.txt`

After finishing the above 3 steps, the applications should be setup locally and ready to run. Please note, you don't have to strictly follow the instructions above. Free to follow the workflow you usually follow in setting up the application.

## Running the Application:

1. The server side of the application can be run by running the **serve.py**, This will start a flask server running on port 5000 on your local system.

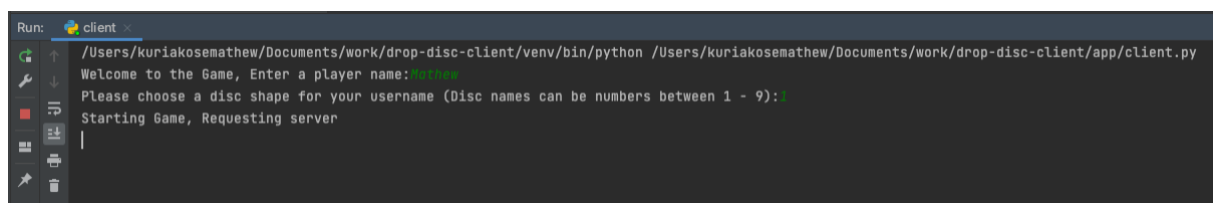


```
run: serve
/usr/local/bin/python3.9 /Users/kuriakosemathew/Documents/work/drop-disc-server/app/serve.py

Application Context not yet initialized, Initializing the Context
* Serving Flask app 'app.controller' (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: off
INFO:werkzeug: * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Fig: 1

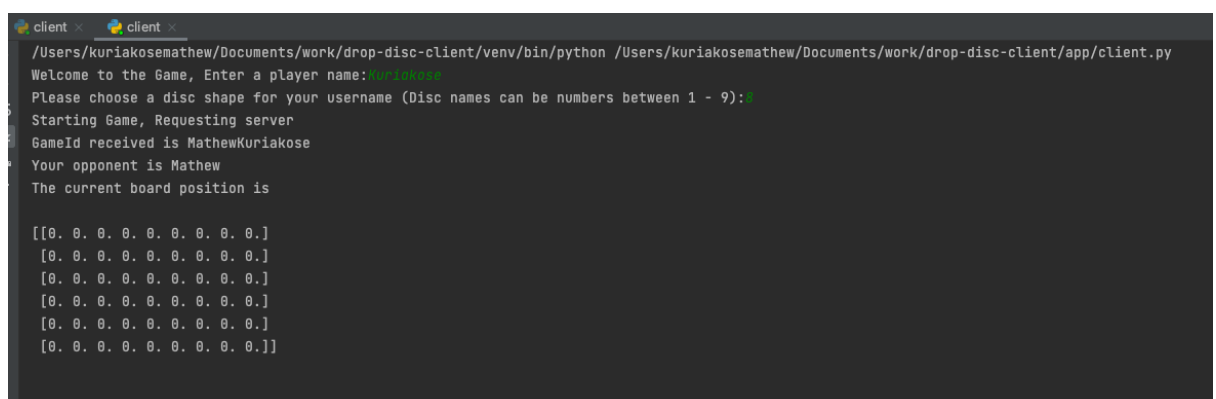
2. The server should be running and will start accepting player request if you see messages like the fig. 1
3. The client side of the application can run by running the **client.py**, This will start a **single instance** of the client. The application will prompt to have a userid and token shape to play the game.



```
Run: client
/Users/kuriakosemathew/Documents/work/drop-disc-client/venv/bin/python /Users/kuriakosemathew/Documents/work/drop-disc-client/app/client.py
Welcome to the Game, Enter a player name: Mathew
Please choose a disc shape for your username (Disc names can be numbers between 1 - 9):
Starting Game, Requesting server
```

Fig: 2

4. Fig 2, the expected state after launching a single instance of client. The client has requested the server and is now waiting for a second player to start the game.
5. Run another instance of client to act as a player 2.



```
client client
/Users/kuriakosemathew/Documents/work/drop-disc-client/venv/bin/python /Users/kuriakosemathew/Documents/work/drop-disc-client/app/client.py
Welcome to the Game, Enter a player name: Mathew
Please choose a disc shape for your username (Disc names can be numbers between 1 - 9):
Starting Game, Requesting server
GameId received is MathewKuriakose
Your opponent is Mathew
The current board position is

[[0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

Fig 3:

6. Figure 3 shows a second player which connected to the game, Since we had a player waiting already in the queue, they got matched and the initial board and opponent details are displayed as can be seen above.

7. The game starts with one of the player being prompted to enter a col number to make the move, while the other player checks the server to get his turn to make the move.
8. Once player1 makes the move, the server updates the next\_player\_turn to player2, The player2 will get notified of its turn when it makes a request to the server.
9. The game alters between player 1 and player2 and updating the board position, until one makes a winning move and emerges as the winner.
10. Attaching few screenshot of the game in action.

```

/Users/kuriakosemathew/Documents/work/drop-disc-client/venv/bin/python /Users/kuriakosemath
Welcome to the Game, Enter a player name:Mathew
Please choose a disc shape for your username (Disc names can be numbers between 1 - 9):1
Starting Game, Requesting server
GameId received is MathewKuriakose
Your opponent is Mathew
The current board position is

[[0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]]

Current Status of the board

[[0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]]

It's your turn Mathew, Please enter a column (1 - 9):1

```

Player1 terminal

```

Welcome to the Game, Enter a player name:kuriakose
Please choose a disc shape for your username (Disc names can be numbers between 1 - 9):
Starting Game, Requesting server
GameId received is MathewKuriakose
Your opponent is Mathew
The current board position is

[[0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]]

Current Status of the board

[[0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0.]]

It's your turn Kuriakose, Please enter a column (1 - 9):

```

Player 2 terminal

```

It's your turn Mathew, Please enter a column (1 - 9):
[[0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 8. 0. 0. 0. 0.]]

```

Player1 terminal: Note player2 made a move, and column 5 stores the disc (number 8) of player 2.

11. The game continues until a winning move is found, the winner flag is updated in the server game state and the clients exit when a winner is found displaying the winner name.

```

It's your turn Mathew, Please enter a column (1 - 9):
[[0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 8. 8. 8. 8. 0.]]

GAME OVER, WINNER IS Mathew

Process finished with exit code 0

```

Player 1 terminal

```

It's your turn Kuriakose, Please enter a column (1 - 9):
[[0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 8. 8. 8. 8. 0.]]

GAME OVER, WINNER IS Mathew

```

Player2 terminal

## Important Implementation Details

### REST API Reference

#### 1. {baseEndpoint}/startgame/

Accepts player details and puts players in a player queue. Takes care of matching players and starting the game.

#### 2. {baseEndpoint}/nextmove/

Accepts move from the player and updates the game board.

#### 3. {baseEndpoint}/getgamestate/

Accepts a gameid & returns the current gamestate of the game.

### Server Implementation Details

1. A Singleton 'ApplicationContext' class powers the server side of the drop disc game. The applicationcontext holds the playerpool queue and the games in game queue. This object will be used across the application to manage the players and game.
2. The singleton object along with the usage of thread-safe data structures and operations are used to manage the states of the game and application.
3. The Server contains the following 3 main classes to handle the state of the game:
  - a. Board:
    - i. Holds the current state of the board, The board is initialized as a numpy array with all zeros at start.

- ii. The board class also contains all operations related to updating the board
- b. Game:
  - i. The Game Object contains players (player 1 & player 2) and the current Board object in play between the players.
  - ii. A unique gameid identifies the game object along with the players and the board associated with the game object.
- c. Player
  - i. Stores the player name and player token.

### Client Implementation Details

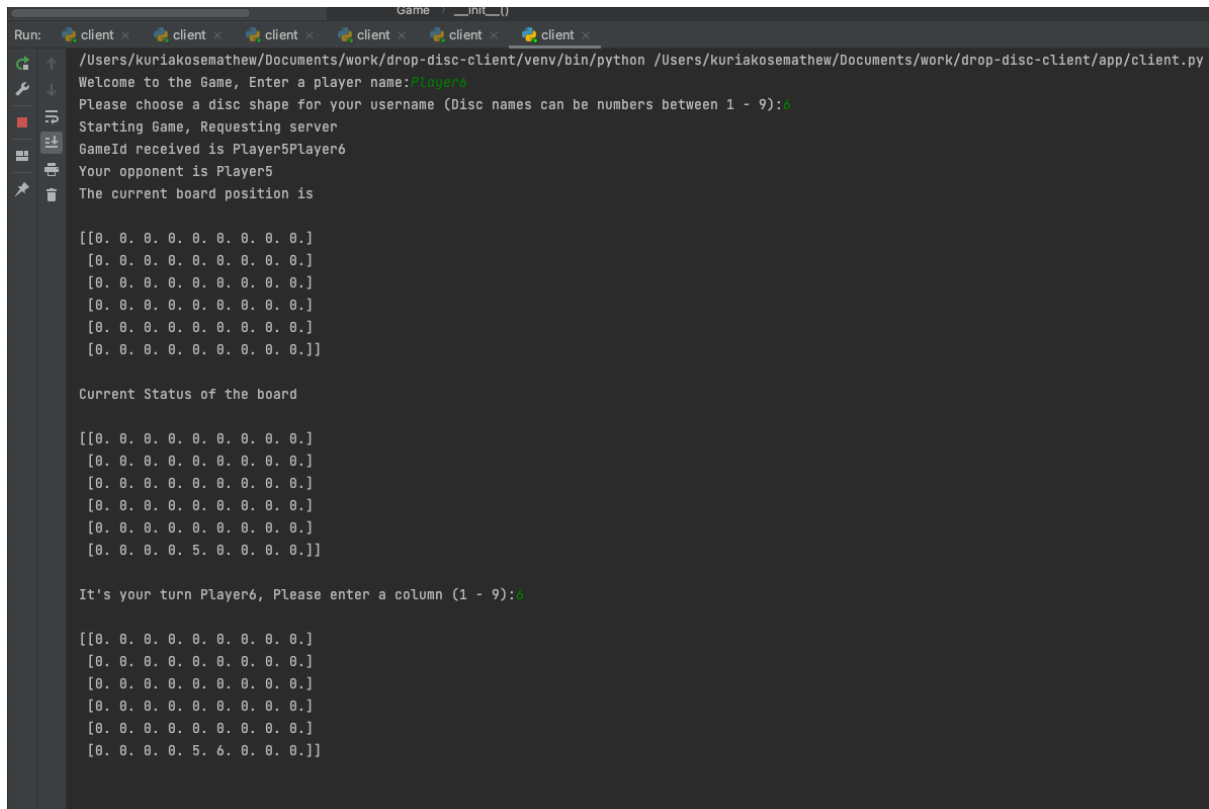
1. The Client is a simple application which contains the following object to manage the state:
  - a. Player: To capture and store player details.
  - b. Game: To capture the current state of the game as it progresses.
2. The core of the client is a simple while loop, which prompts the server on the status of the game. When the turn comes, it prompts the user to make an input otherwise it will request the server for a status every other second.
3. Exits when a winner is found on server request.

### Important Design Considerations

1. The state is managed within the server itself as using a db/filesystem leads to additional requests on server from clients to match opponents. Currently, the only additional requests that comes from the client is to check their turn.
2. Python was used as the language of choice, purely based on the developer familiarity with python, flask and numpy. For production purposes, Java is preferred.
3. Flask was used as the web service development choice due to familiarity.
4. Numpy was used underneath, for abstracting away all the difficulties in managing a custom array. The current version uses a number data type, This decision was made purely from aesthetic point of view. The number data type looks prettier on the screen and increases the ux of players playing. This can be easily extended to support characters by writing a display function which prettifies the numpy array.
5. Configuration is managed using the application.conf file (Committed inside the repo).
6. Loggers are implemented in server and current logging is to stdout. The loggers in client are written to a file inside the application. This choice was done as the stdout is used for gameplay. Future accommodation can be made to upload this log file to server after a crash.
7. Thread safe operations of Queue & dictionary are used to manage state.
8. Healthcheck apis are implemented in server side, as the application will be running in dockers over orchestration servies and healthcheck will be important.
9. The assignment of gameid & player matching are simple implementations developed in the interest of time. Better algorithms need to be implemented before this can be moved to a more production like environment.

## Testing

1. Unit tests are written in server side to test core game logic. Check test module in server repo.
2. **Concurrent playing.** The game was played with 6 different instance of clients(players) playing 3 different games concurrently without issues.



```
Run: client x client x client x client x client x client x
/Users/kuriakosemathew/Documents/work/drop-disc-client/venv/bin/python /Users/kuriakosemathew/Documents/work/drop-disc-client/app/client.py
Welcome to the Game, Enter a player name: Player6
Please choose a disc shape for your username (Disc names can be numbers between 1 - 9):
Starting Game, Requesting server
GameId received is Player5Player6
Your opponent is Player5
The current board position is

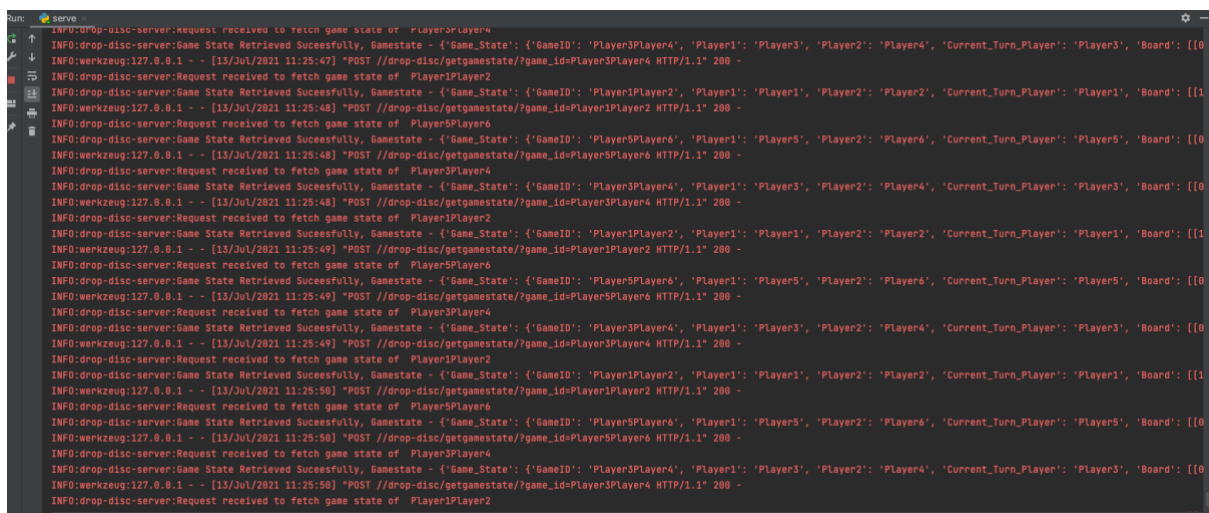
[[0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]]

Current Status of the board

[[0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 5. 0. 0. 0. 0.]]

It's your turn Player6, Please enter a column (1 - 9):
[[0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 5. 6. 0. 0. 0.]]
```

Fig: 6 client instance running



```
Run: serve
INFO:drop-disc-server:request received to fetch game state of Player3Player4
INFO:drop-disc-server:Game State Retrieved Successfully, Gamestate - {'Game_State': {'GameID': 'Player3Player4', 'Player1': 'Player3', 'Player2': 'Player4', 'Current_Turn_Player': 'Player3', 'Board': [[0
INFO:werkzeug:127.0.0.1 - - [13/Jul/2021 11:25:47] "POST //drop-disc/getgamestate/?game_id=Player3Player4 HTTP/1.1" 200 -
INFO:drop-disc-server:request received to fetch game state of Player1Player2
INFO:drop-disc-server:Game State Retrieved Successfully, Gamestate - {'Game_State': {'GameID': 'Player1Player2', 'Player1': 'Player1', 'Player2': 'Player2', 'Current_Turn_Player': 'Player1', 'Board': [[1
INFO:werkzeug:127.0.0.1 - - [13/Jul/2021 11:25:48] "POST //drop-disc/getgamestate/?game_id=Player1Player2 HTTP/1.1" 200 -
INFO:drop-disc-server:request received to fetch game state of Player5Player6
INFO:drop-disc-server:Game State Retrieved Successfully, Gamestate - {'Game_State': {'GameID': 'Player5Player6', 'Player1': 'Player5', 'Player2': 'Player6', 'Current_Turn_Player': 'Player5', 'Board': [[0
INFO:werkzeug:127.0.0.1 - - [13/Jul/2021 11:25:48] "POST //drop-disc/getgamestate/?game_id=Player5Player6 HTTP/1.1" 200 -
INFO:drop-disc-server:request received to fetch game state of Player3Player4
INFO:drop-disc-server:Game State Retrieved Successfully, Gamestate - {'Game_State': {'GameID': 'Player3Player4', 'Player1': 'Player3', 'Player2': 'Player4', 'Current_Turn_Player': 'Player3', 'Board': [[0
INFO:werkzeug:127.0.0.1 - - [13/Jul/2021 11:25:48] "POST //drop-disc/getgamestate/?game_id=Player3Player4 HTTP/1.1" 200 -
INFO:drop-disc-server:request received to fetch game state of Player1Player2
INFO:drop-disc-server:Game State Retrieved Successfully, Gamestate - {'Game_State': {'GameID': 'Player1Player2', 'Player1': 'Player1', 'Player2': 'Player2', 'Current_Turn_Player': 'Player1', 'Board': [[1
INFO:werkzeug:127.0.0.1 - - [13/Jul/2021 11:25:49] "POST //drop-disc/getgamestate/?game_id=Player1Player2 HTTP/1.1" 200 -
INFO:drop-disc-server:request received to fetch game state of Player5Player6
INFO:drop-disc-server:Game State Retrieved Successfully, Gamestate - {'Game_State': {'GameID': 'Player5Player6', 'Player1': 'Player5', 'Player2': 'Player6', 'Current_Turn_Player': 'Player5', 'Board': [[0
INFO:werkzeug:127.0.0.1 - - [13/Jul/2021 11:25:49] "POST //drop-disc/getgamestate/?game_id=Player5Player6 HTTP/1.1" 200 -
INFO:drop-disc-server:request received to fetch game state of Player3Player4
INFO:drop-disc-server:Game State Retrieved Successfully, Gamestate - {'Game_State': {'GameID': 'Player3Player4', 'Player1': 'Player3', 'Player2': 'Player4', 'Current_Turn_Player': 'Player3', 'Board': [[0
INFO:werkzeug:127.0.0.1 - - [13/Jul/2021 11:25:49] "POST //drop-disc/getgamestate/?game_id=Player3Player4 HTTP/1.1" 200 -
INFO:drop-disc-server:request received to fetch game state of Player1Player2
INFO:drop-disc-server:Game State Retrieved Successfully, Gamestate - {'Game_State': {'GameID': 'Player1Player2', 'Player1': 'Player1', 'Player2': 'Player2', 'Current_Turn_Player': 'Player1', 'Board': [[1
INFO:werkzeug:127.0.0.1 - - [13/Jul/2021 11:25:50] "POST //drop-disc/getgamestate/?game_id=Player1Player2 HTTP/1.1" 200 -
INFO:drop-disc-server:request received to fetch game state of Player5Player6
INFO:drop-disc-server:Game State Retrieved Successfully, Gamestate - {'Game_State': {'GameID': 'Player5Player6', 'Player1': 'Player5', 'Player2': 'Player6', 'Current_Turn_Player': 'Player5', 'Board': [[0
INFO:werkzeug:127.0.0.1 - - [13/Jul/2021 11:25:50] "POST //drop-disc/getgamestate/?game_id=Player5Player6 HTTP/1.1" 200 -
INFO:drop-disc-server:request received to fetch game state of Player3Player4
INFO:drop-disc-server:Game State Retrieved Successfully, Gamestate - {'Game_State': {'GameID': 'Player3Player4', 'Player1': 'Player3', 'Player2': 'Player4', 'Current_Turn_Player': 'Player3', 'Board': [[0
INFO:werkzeug:127.0.0.1 - - [13/Jul/2021 11:25:50] "POST //drop-disc/getgamestate/?game_id=Player3Player4 HTTP/1.1" 200 -
INFO:drop-disc-server:request received to fetch game state of Player1Player2
INFO:drop-disc-server:Game State Retrieved Successfully, Gamestate - {'Game_State': {'GameID': 'Player1Player2', 'Player1': 'Player1', 'Player2': 'Player2', 'Current_Turn_Player': 'Player1', 'Board': [[1
INFO:werkzeug:127.0.0.1 - - [13/Jul/2021 11:25:50] "POST //drop-disc/getgamestate/?game_id=Player1Player2 HTTP/1.1" 200 -
```

Fig 7: Server logs when 3 different games where running simultaneously. Note the gameId which identifies the 3 different games.

### Areas to Improve:

1. Better player matching algorithm.
2. Bi-directional communication to reduce load on server.
3. Better assignment of gameid to handle games in production well.
4. Improved logs.
5. Better exception handling, the current ones in place are really basic developed in the interest of time.
6. Containerisation of application.

### Path to Production

A simple architecture of how to take to production:

