**Group members:**

* Nikola Dordevic, s341839
* Jørund Topp Løvlien, s341822

**Running the program:**client.py  
server.py

The assignment is programmed in Python 3.9.2  
Libraries needed for the client and server to work are gRPC and PyGame  
To install them run:

pip install grpcio

pip install grpcio-tools

pip install pygame

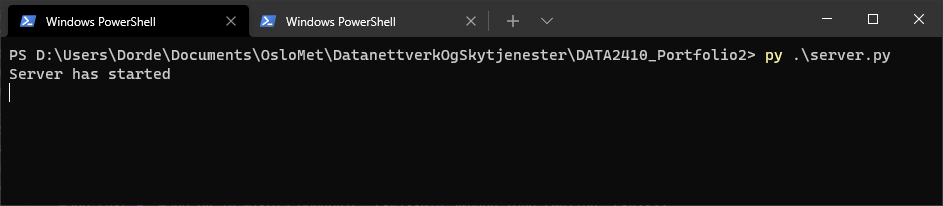
To get started you first run the server this can be done through a dockerfile or manually through the terminal. The server has to be running for to play the game, singleplayer or multiplayer, else the client will be prompted an error message.

Afterwards you can run the game as a client. The game supports endless amounts of players.

**Example run:**NOTE: Each command is run on a separate terminal.

Runiing the game…

**Example images of run:**

**Server:**

**Client:**

Figure : Start menu

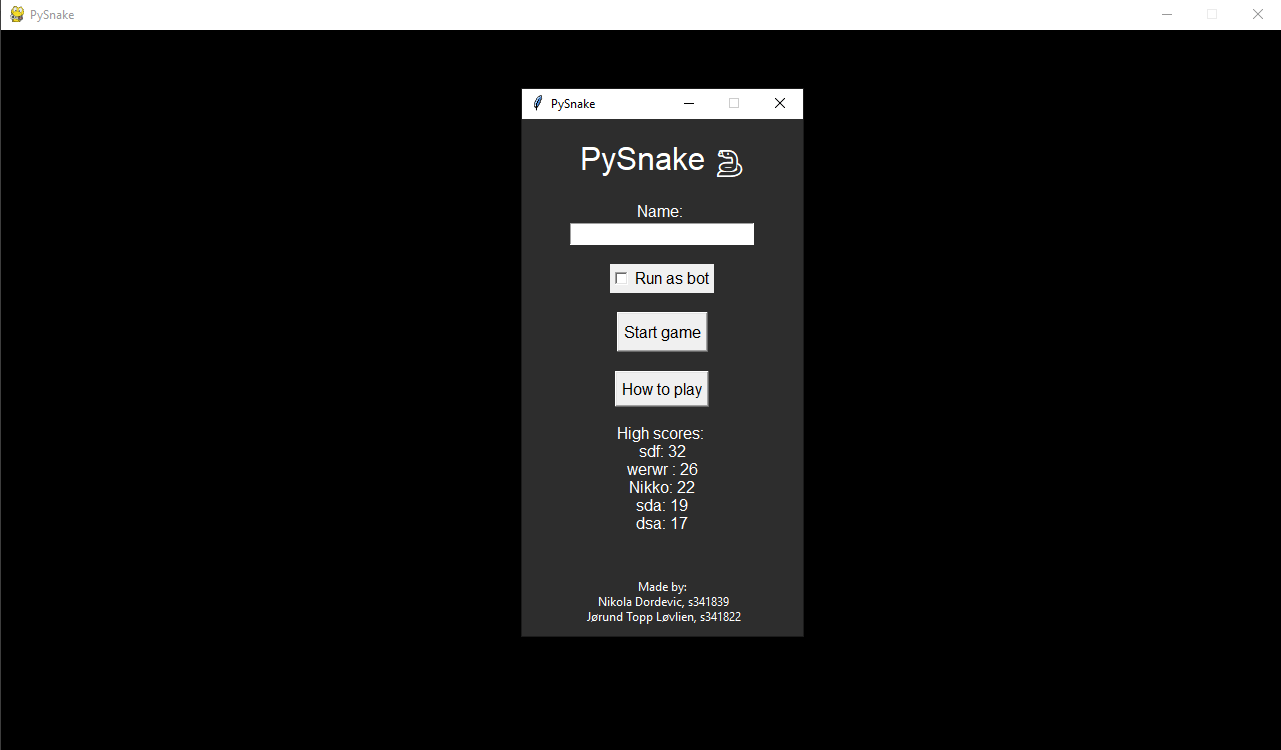
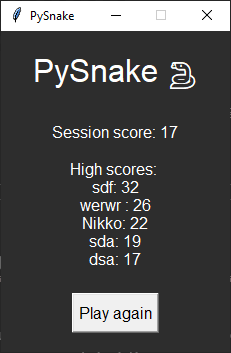
****

Figure : Whilst in a game session

****

Figure : End of a session



**Bots:**The game supports bots. To play as a bot you must tick off a checkbox on the menu titled: “Run as bot”. Once that is done, input the name and start the game as normal.

**Algorithm**  
The bots use the A\* search algorithm.  
Firstly it has information on all the fruits on the board. It compares each fruit by the distance between its head and the fruit:

abs(p1[0] - p2[0]) + abs(p1[1] - p2[1])

Where p stands for point, and [0] for x, [1] for y, in a point (x, y)  
After it has chosen the closest fruit possible it starts calculating the path.

The bot treats every tile on the board as a node, using a linked list. Similarly, to Dijkstra’s Algorithm it compares each path possible by the weight of the distance. However, since each node is part of a grid, we can assume that the weight is exactly 1. Using this knowledge, we compare each neighbor to the snake head using this formula:  
 f(n) = g(n) + h(n)  
g(n) is the weight between the node and the head, while h(n) is the distance between the new point and the fruit closest to the snake head. What this does is, if the neighboring node we are testing is further away from the point than the other neighboring nodes, we can safely throw it away knowing that it does not contain the fastest path. After finding the closest node, first it assigns the node we came from as a parent then it recursively checks its neighbors until finally reaching the fruit. Finally, we get the path by putting each node’s coordinates into an array and iterating through the linked list via their parents.