Relatório de Implementação

Projeto da cadeira de Programação II – dronyDeliv

- Grupo 82
- José Miguel Vilhena de Almeida – nº 55373
- Miguel Lourenço Lages – nº 54975

Creditação:

José:

- organize.py
- timeTools.py
- Combo.py
- DronyD.py
- AutoDronyD.py
- FileNames.py

Miguel:

- readFiles.py
- writeFiles.py
- Drone.py
- Parcel.py
- Header.py

Ambos tratamos do dronyD.py e fomos sempre conversando por videochamada à medida que avançávamos na criação e resolução do programa, explicando tudo o que fizemos um ao outro de forma a que cada um de nós não só entendesse o que o outro fez, mas também o código do programa em si.

Funções implementadas:

- **dronyD.py** – permite executar o programa chamando os dois ficheiros (o ficheiro de drones e o das encomendas) na linha de comandos (consola), seguindo a seguinte sintaxe: **python dronyD.py <ficheiro de drone> <ficheiro de listagem de encomendas>**

- **autoDronyD.py** – uma função adicional que permite executar o programa sem ser necessário chamar os dois ficheiros na linha de comandos (consola). Apenas é necessário que os dois ficheiros (o ficheiro de drones e o das encomendas) se encontrem no mesmo diretório que o ficheiro *autoDrony.py*.

- **readFiles.py** – recebe os ficheiros de drones e de encomendas e coloca as informações necessárias nos respetivos objetos. Ou seja, recebe a informação dos ficheiros com os drones e coloca cada linha (drone) num objeto *Drone*, com os respetivos atributos; recebe a informação dos ficheiros das encomendas e coloca em objetos *Parcel*; e recebe a informação do cabeçalho e coloca em objetos *Header*.

- **Drone.py** – classe criadora de objetos que representam drones, onde se encontram todos os atributos recebidos no *inputFile* que foi submetido. Contém também os método de acesso (getters) e o métodos de mudança de estado (setters).

- **Parcel.py** – classe criadora de objetos que representam encomendas, onde se encontram todos os atributos recebidos no *inputFile* que foi submetido. Contém também os métodos de acesso (getters) e os métodos de mudança de estado (setters).

- **Header.py -** classe criadora de objetos que representam cabeçalhos, onde se encontram todos os atributos recebidos no cabeçalho do *inputFile* que foi submetido. Contém também os métodos de acesso (getter).

- **Combo.py** – classe implementada para associar **drones** ás *respetivas* **parcels**.. Define também o estado das encomendas.

- **organize.py** – define qual o melhor **drone** para cada **encomenda**, seguindo os vários critérios necessários.

- **writeFiles.py** – recebendo os dados de outras funções alojadas no readFiles.py e organize.py, cria os ficheiros de saída com nomes adequados e escreve os dados recebidos corretamente nos novos ficheiros.

- **timeTools.py** – contêm um conjunto de funções utilizadas para vários fins, como por exemplo: converter *timestamps* em objetos *datetime*, atualizar *timestamps*, entre outros.

- **FileNames.py** – uma classe que permite criar objetos alojadores de nomes de ficheiros. Completa com métodos de acesso (getters) para aceder a tais nomes de ficheiros alojados nos objetos.

Funções por implementar:

- Da nossa perspetiva não ficou nenhuma função por implementar.

Erros conhecidos:

- Não foram detetados nenhuns erros durante realização do projeto, nem durante a execução do programa.