https://github.com/mitchpk/oopmajorpractical

Transport delivery business game
- Which fuel to use
- Vehicle types with different stats
- Different vehicles have different prices
- Further distances with higher profit but more risk
- Random events along the way

Create class diagrams with inheritance and virtual classes

The story is: you are running a delivery company and you need to choose the best options to deliver x packages. You can choose between air, land or sea vehicles and each has its benefits and drawbacks.

```
class TravelResult:
        bool failed
        int distanceTravelled

class Vehicle:
        virtual travel() -> TravelResult
        unsigned int speed
        float fuelRate
        std::vector<Package> loadedPackages

class Truck:
        travel() override -> TravelResult

class Aeroplane:
        travel() override -> TravelResult

class Ship:
        travel() override -> TravelResult

class Package:
        unsigned int weight
        string description

map of city names to
        struct Destination:
                unsigned int distance
                accessibleBy // aircraft, truck and/or ship
```

Project Specification - Major Practical

Mitchell Kempen - a1827437 | Zane Pattinson - a1803664 | Antony - a1822479

Introduction

Our project implements a transport delivery business game. The player first selects what delivery job they would like to do which gives them a required distance to travel. They then select which vehicle they would like to use, which could be a plane, boat or truck, each with different stats, prices, benefits and drawbacks. The player then chooses what fuel they would like to use and how much of it, trying to maximise distance with the lowest cost. Once the player has confirmed their choices the simulation begins with their chosen vehicle travelling to the destination. Along the way there may be random events that the player has to respond to. If the player successfully makes it to their destination without running out of fuel, breaking down or losing for some other reason they are rewarded with an amount of money.

Design Description

Memory allocation from the stack and the heap
- Arrays: <how we will use this aspect>
- Strings: <how we will use this aspect>
- Objects: <how we will use this aspect>

User Input and Output
- I/O of different data types: <how we will use this aspect>

Object-oriented programming and design
- Inheritance: <how we will use this aspect>
- Polymorphism: <how we will use this aspect>
- Abstract Classes: <how we will use this aspect>

Testing
- Test Inputs / Expected Outputs: <how we will use this aspect>
- Automated Testing: <how we will use this aspect>
- Regression Testing: <how we will use this aspect>

Class Diagram

<insert class diagram that shows how classes connect together>

Class Descriptions

Vehicle class:
Base class which all vehicles inherit from. Contains a speed, max weight and fuel type which influences how far the vehicle can travel. There is a virtual function called travel which takes a distance in kilometres that the user provides, being a portion of the entire trip length.

e.g. shipment from Adelaide to Melbourne, 800km
Some trips require certain types of vehicles, for example an overseas trip requires an aircraft, whereas a boat is required for a trip to an island

For each shipment, the user chooses one form of transport (one vehicle), purchases a certain amount of fuel, and loads

Company class:
Represents the player's company, including their current balance, owned vehicles, and name.

Package class:
Represents a single package, including weight, description and intended destination. Both the vehicle class and company class store multiple instances of this class.

Code Style
All code in the program will be properly indented using 4 spaces per tab. Classes will begin with a capital letter, in the form Class1.h and Class1.cpp. Function comments will be given for each function, describing what the function does, what the return type represents and what arguments are available for the function. Code comments will be used to describe what each block of code performs if it is not obvious from just reading the code or the code is short. Comments will be written as the code is written.

Testing Plan

Unit Testing
All classes must be tested using a tests folder. When creating a new unit test, add it to the Makefile in order for it to be automatically compiled when running 'make test'

Schedule Plan

Objectives (organised chronologically)
Create makefile and project structure
Set up testing files
Create base vehicle class
Create company class
Create package class
Create basic game loop in main.cpp
Make some preset vehicles (i.e. ute, semi, cruiseliner)

Implement loading packages into vehicle
Implement receiving a reward for successful deliveries
(Beyond this point, these features are things that can be added but aren't necessary)
Create a store class which sells vehicles
Implement the ability to send multiple packages per trip to different destinations

WRITE COMMENTS ALONG THE WAY