Driverless Car: UML Models Reflection

The initial class diagrams were modified as a result of the implementation of system. When designing the system at a high level, not all of the processes and actions were fully described. As a result of the software coding and development, some of the class structures were modified in order to achieve the desired functionality.

The Vehicle Class:

The table below shows the vehicle class at design and implementation phases. Additions are marked in blue and deletions in red:

Design	Implementation
Vehicle reg: string make : string model: string drive: Movement --- locate_road() detect_obstacle() get_speed_limit(road) get_stopping_distance() steer() accelerate() brake()	**Vehicle** cat: type reg: string make : string model: string route: Road log: Log drive: Movement sensor: Sensor --- steer() accelerate() cruise () brake() emergency_brake() update_speed(t) update_location(s,t) distance_travelled(t) drive_car(t) at_checkpoint() emergency_stop()

Upon implementation it became obvious that the car must have a route attribute to know where to go, a sensor to trigger the emergency stop and a log of the journey for recording and testing. The drive method was also introduced which then called upon the component methods to drive the car. The category of vehicle was also recorded.

The Movement Class:

Design	Implementation
Movement location : Point direction: Direction speed: float acceleration: float time: integer	**Movement** location : Point direction: Direction gradient: float speed: float acceleration: float time: integer

It was helpful to split the direction into a 2D movement and gradient value. This made the coding more readable when tackling the change in acceleration due to the slope of the road.

The Sensor Class:

Sensor
emergency_stop : Boolean

The sensor class was introduced to trigger the emergency stop. While this could have been a simple Boolean attribute to the vehicle class, the separate sensor class allows for additional sensors to be added to trigger different actions. This could be for maintaining stopping distance, changing lanes etc.

The Point and Direction Classes:

The Point and Direction classes remained unchanged.

The Road Class:

The get_road_segment method was necessary to construct a route from a list of road segments. The get_speed_limit method was introduced as the speed limit is associated with the road segment rather than the road itself or the vehicle.

Road Segment Class:

Design	Implementation
Road_Segment index: integer location_start : Point location_end:Point direction:Direction distance: int gradient: int category: string ——— get_distance() get_category(road) get_direction() get_gradient()	**Road_Segment** index: integer location_start : Point location_end:Point direction:Direction distance: int gradient: int category: string ——— get_distance() get_direction() get_gradient()

While the get_direction method is shown in the class diagram, it was called as a method from the Point class which is an example of polymorphism.

Reflection

Some of the features missed in the design phase were as a result of a lack of experience with UML modelling. However, the more complex the actions within the system the more likely it is that the class diagrams from the design phase will be altered as a result of the programming and technicalities of the code. In addition, it may become apparent when implementing a problem that more elegant solutions exist. Ultimately the UML is a model to enhance communication in the planning phase, which should be modified in the implementation phase and documented for maintenance.

References

Reddy, P. P. (2019) Driverless Car: Software Modelling and Design using Python and Tensorflow.

Zhou, Z. Q. & Sun, L. (2019) Metamorphic testing of driverless cars. Commun. ACM 62(3): 61–67. DOI: https://doi.org/10.1145/3241979.

Joque, J. (2016) The Invention of the Object: Object Orientation and the Philosophical Development of Programming Languages. *Philosophy & Technology* (29):335 -356.

Rumbaugh, J., Jacobson, I. & Booch, G. (2005) *The Unified Moddeling Language Reference Manual. Second Edition.* Boston: Addison-Wesley

Phillips D. (2018) *Python 3 Object-Oriented Programming*. Third Edition. Birmingham: Packt Publishing Ltd.

Glinz, M. (2000) 'Problems and deficiencies of UML as a requirements specification language', *Tenth International Workshop on Software Specification and Design.* San Diego, 07 November 2000. San Diego: IEEE. 11-22.

Babaei, P et al. (2023) Perception System Architecture for Self-Driving Vehicles: A Cyber- Physical Systems Framework. Available from: https://www.researchsquare.com/article/rs-3777591/v1 [Accessed 25 April 2024]