Princess Power

COS 214 Group Project

INITIAL DESIGN

Team manager : Mpendulo Sibeko

Group Number : 21

Group Members :

Claire Smith	U19185032
Dharshan Moodley	U19072164
Matthew Reed	U19100133
Matthew Schoeman	U17029377
Mayur Pillay	U19290498

GitHub repository : Click here

Google Doc:
https://docs.google.com/document/d/1dk97ZT0ak89xvm_BFQYpzkH-qIExcK5eIW3KRvbn3_o/edit?usp=sharing

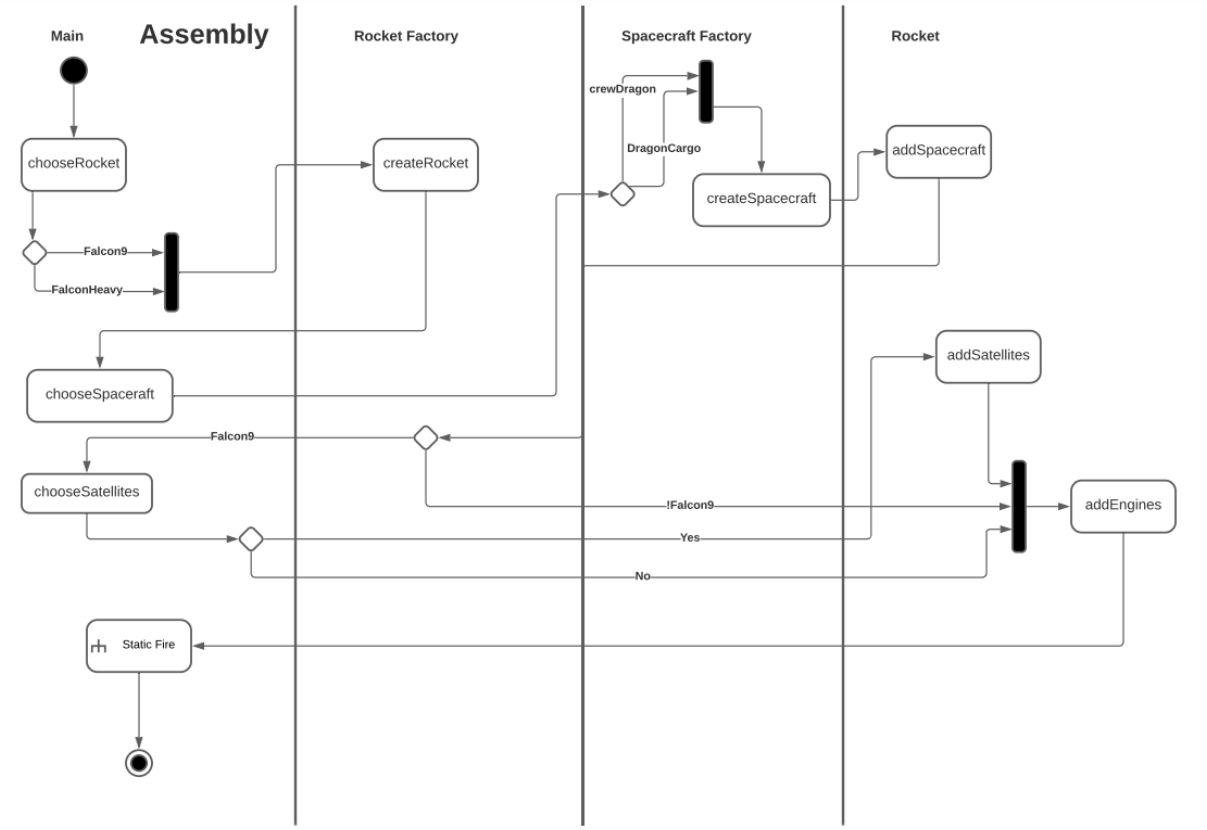
Google Drive:
https://drive.google.com/drive/folders/1Kt9rlesctVdTLTcOFYVIpncMQreCptD_?usp=sharing

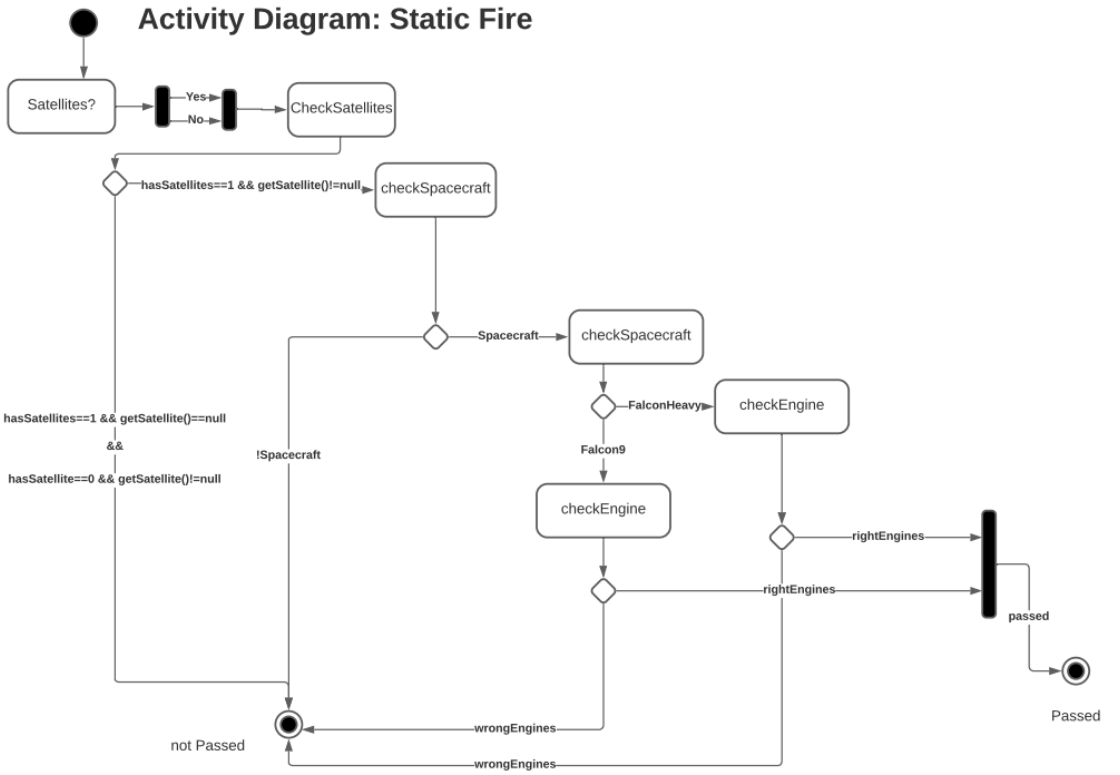
Task 1 :

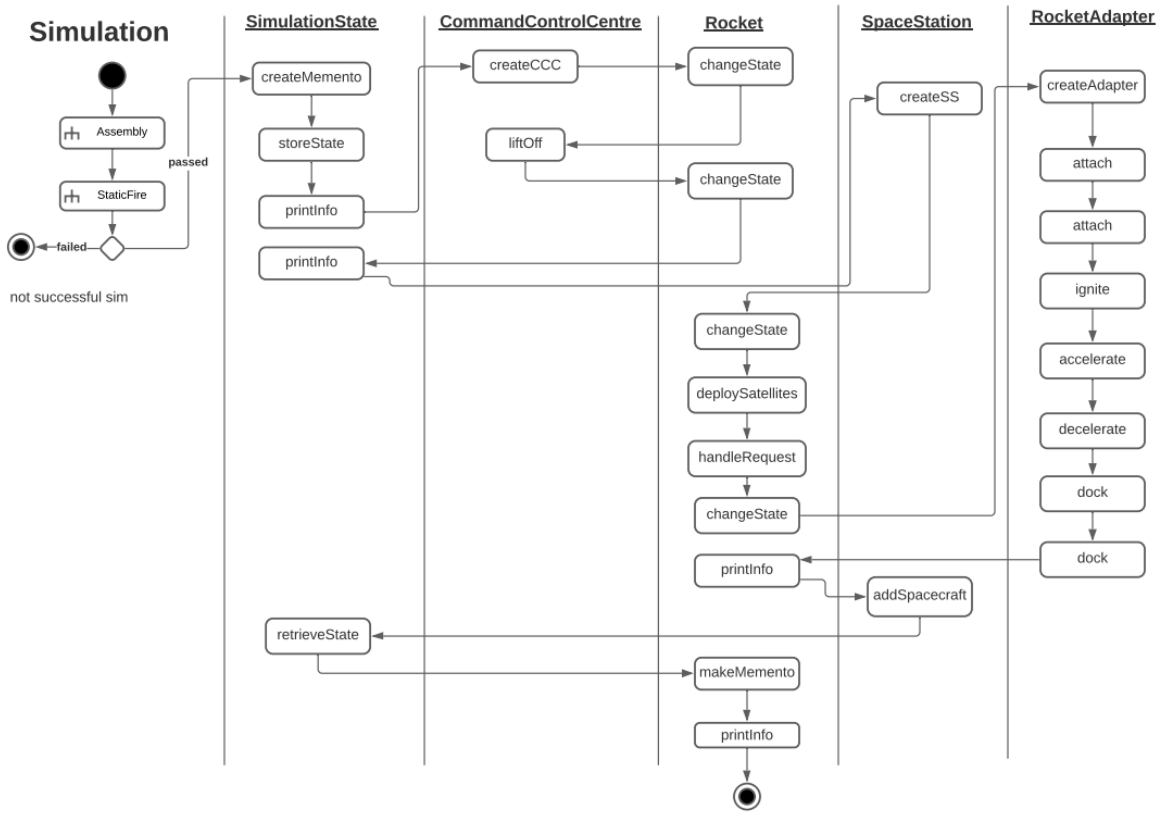
1.1) Identify the functional requirements

The simulation must create two types of rockets, a Falcon 9 rocket and a Falcon Heavy rocket. The Falcon 9 rocket can carry a spacecraft object and a cluster of satellite objects. Before a rocket launches it must do a static fire test. The simulation must allow for user input to define what type of rocket and how many satellites.

1.2) https://drive.google.com/file/d/1BeNQUjs6HavwniVeelqLpyFa1MuHMJlK/view?usp=sharing




Activity Diagram: Static Fire
Satellites?
Yes
No
CheckSatellites
hasSatellites==1 && getSatellite()!=null
checkSpacecraft
Spacecraft
checkSpacecraft
FalconHeavy
checkEngine
hasSatellites==1 && getSatellite()==null
&&
hasSatellite==0 && getSatellite()!=null
!Spacecraft
Falcon9
checkEngine
rightEngines
rightEngines
passed
Passed
not Passed
wrongEngines
wrongEngines

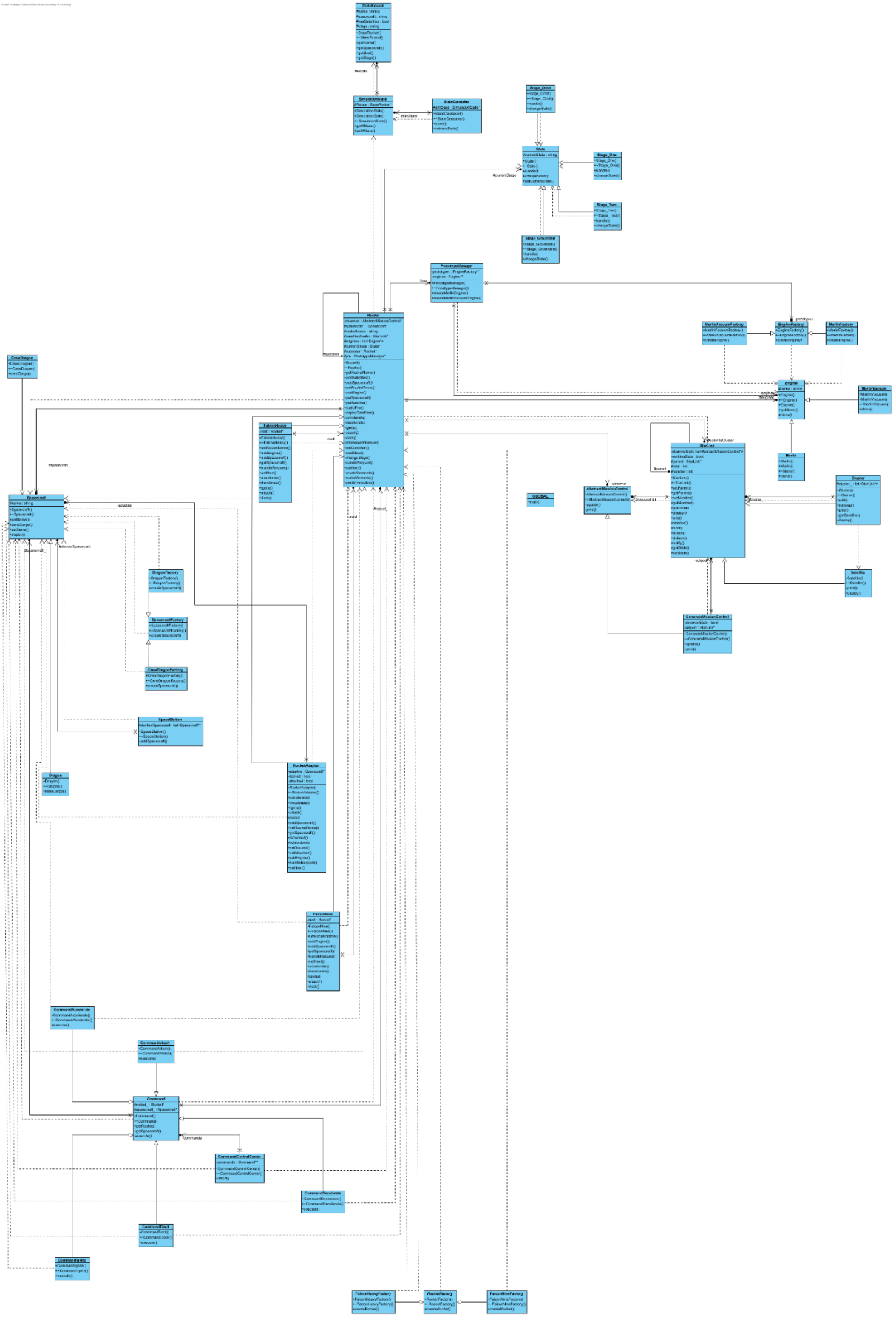

Simulation
SimulationState
CommandControlCentre
Rocket
SpaceStation
RocketAdapter
Assembly
StaticFire
failed
passed
not successful sim
createMemento
storeState
printInfo
printInfo
createCCC
liftOff
changeState
changeState
changeState
deploySatellites
handleRequest
changeState
printInfo
createSS
addSpacecraft
createAdapter
attach
attach
ignite
accelerate
decelerate
dock
dock
retrieveState
makeMemento
printInfo

1.3. Design Patterns	1.4. Class Name (Participant)
Factory	SpacecraftFactory (Creator)
	RocketFactory (Creator)
	CrewDragonFactory (Concrete Creator)
	DragonFactory (Concrete Creator)
	FalconNineFactory(Concrete Creator)
	FalconHeavyFactory(Concrete Creator)
	Rocket (Product)
	FalconNine (Concrete Product)
	FalconHeavy(Concrete Product)
	Spacecraft (Product)
	Dragon (Concrete Product)
	CrewDragon (Concrete Product)
	EngineFactory (Creator)
	MerlinVacuumFactory (Concrete Creator)
	MerlinFactory (Concrete Creator)
	Engine (Product)
	MerlinVacuum (Concrete Product)
	Merlin (Concrete Product)
Prototype	Engine (Prototype)
	Merlin (Concrete Prototype)
	MerlinVacuum (Concrete Prototype)
	PrototypeManager (Client)

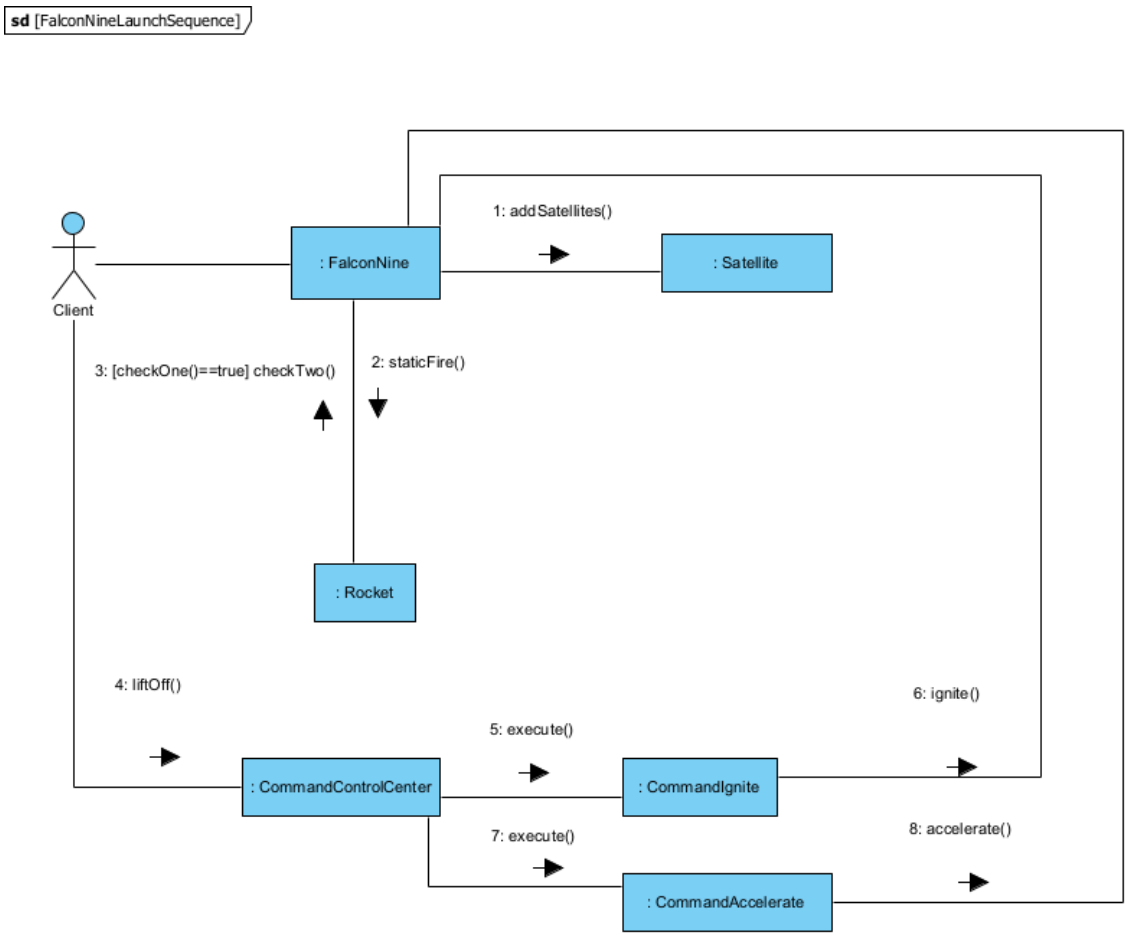
Composite	StarLink (Component)
	Satellite (Leaf)
	Cluster (Composite)
State	State (State)
	Rocket (Context)
	Stage_One (Concrete State)
	Stage_Two (Concrete State)
	Stage_Grounded (Concrete State)
	Stage_Orbit (Concrete State)
Memento	SimulationState (Memento)
	StateCaretaker (Caretaker)
	StateRocket (Originator)
Command	Command (Command)
	CommandIgnite (Concrete Command)
	CommandDecelerate (Concrete Command)
	CommandAccelerate (Concrete Command)
	CommandDock (Concrete Command)
	CommandAttach (Concrete Command)
	CommandControlCenter (Receiver)
	Rocket (Invoker)
	Spacecraft (Invoker)
Template Method	Rocket (Abstract Class)
	FalconNine (Concrete Class)
	FalconHeavy (Concrete Class)

Observer	AbstractMissionControl (Observer)
	ConcreteMissionControl (Concrete Observer)
	Starlink (Subject)
	Satellite (Concrete Subject)
Chain Of Responsibility	Rocket (Handler)
	FalconNine (Concrete Handler)
	FalconHeavy (Concrete Handler)
Adapter	RocketAdapter (Adapter)
	Spacecraft (Adaptee)
	Rocket (Target)

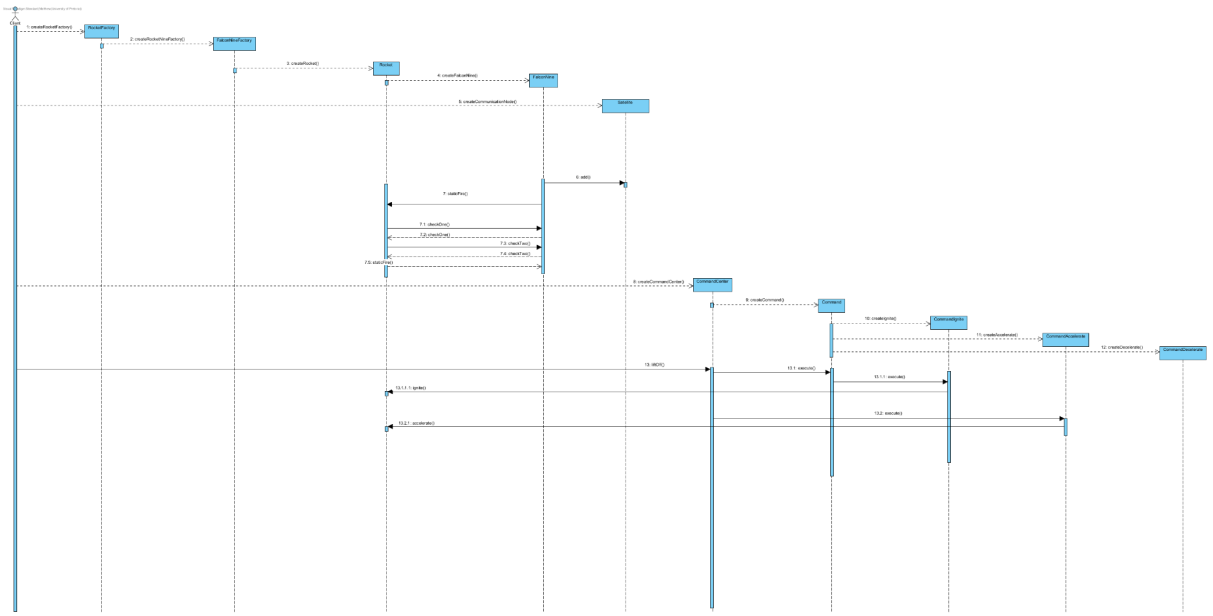
1.5) https://drive.google.com/file/d/1OWW4xFBa2uE2-NjooycKe18f7chaAkq3/view?usp=sharing



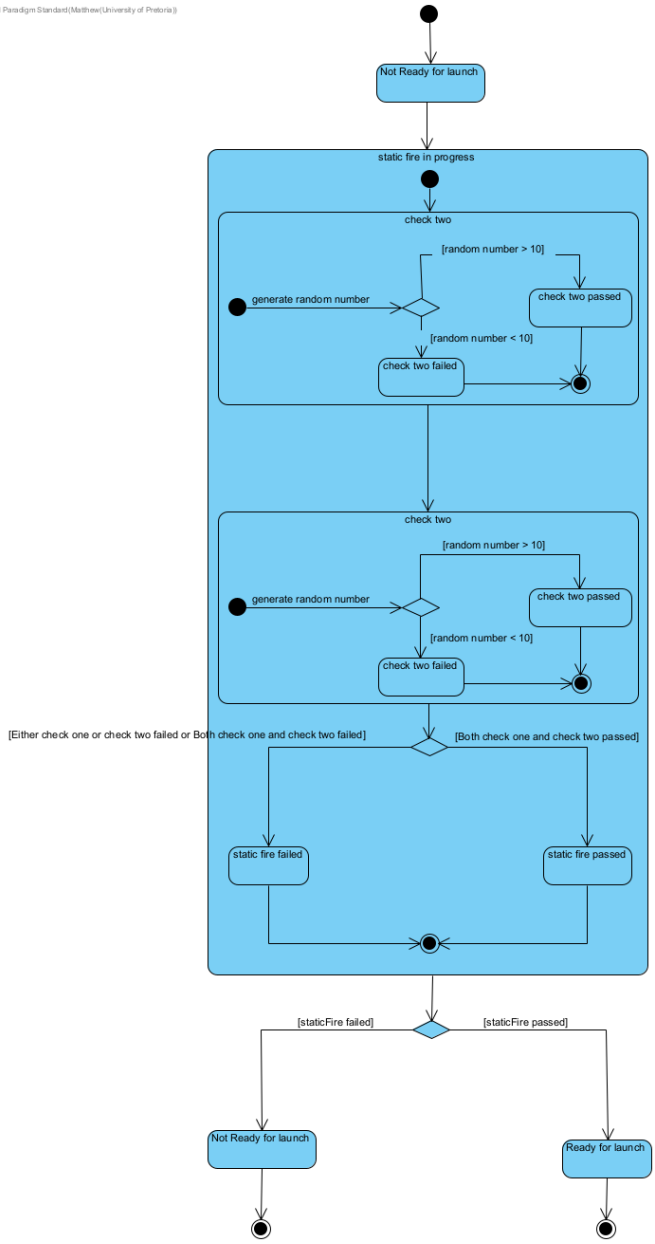
1.6.1)



1.6.2)https://drive.google.com/file/d/1K3DMkmwpzdfKliA8BTOx5dTRmNOW8Aw3/view?usp=sharing



1.7)https://drive.google.com/file/d/1ET12Gz61fc3FJwp_QHmXkZ0wx0iFibQ2/view?usp=sharing

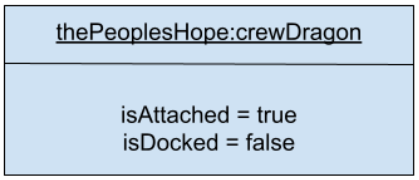
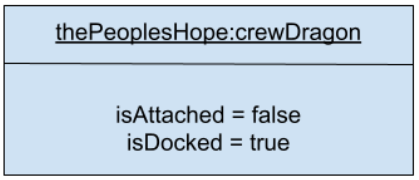
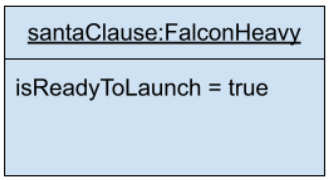
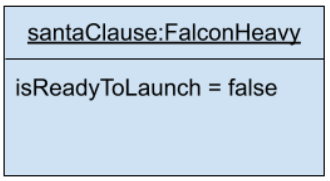


This diagram shows how a rocket changes state from "Not ready to launch" to "Ready to launch."

The rocket object's default state is "Not Ready To Launch"

The state of the rocket object changes when the client calls the staticFire() method. The static fire method has two outcomes which determine whether the Rocket's state changes. If the staticFire method passes then the Rocket's state changes to Ready To Launch. If it fails then the Rocket's state remains the same.

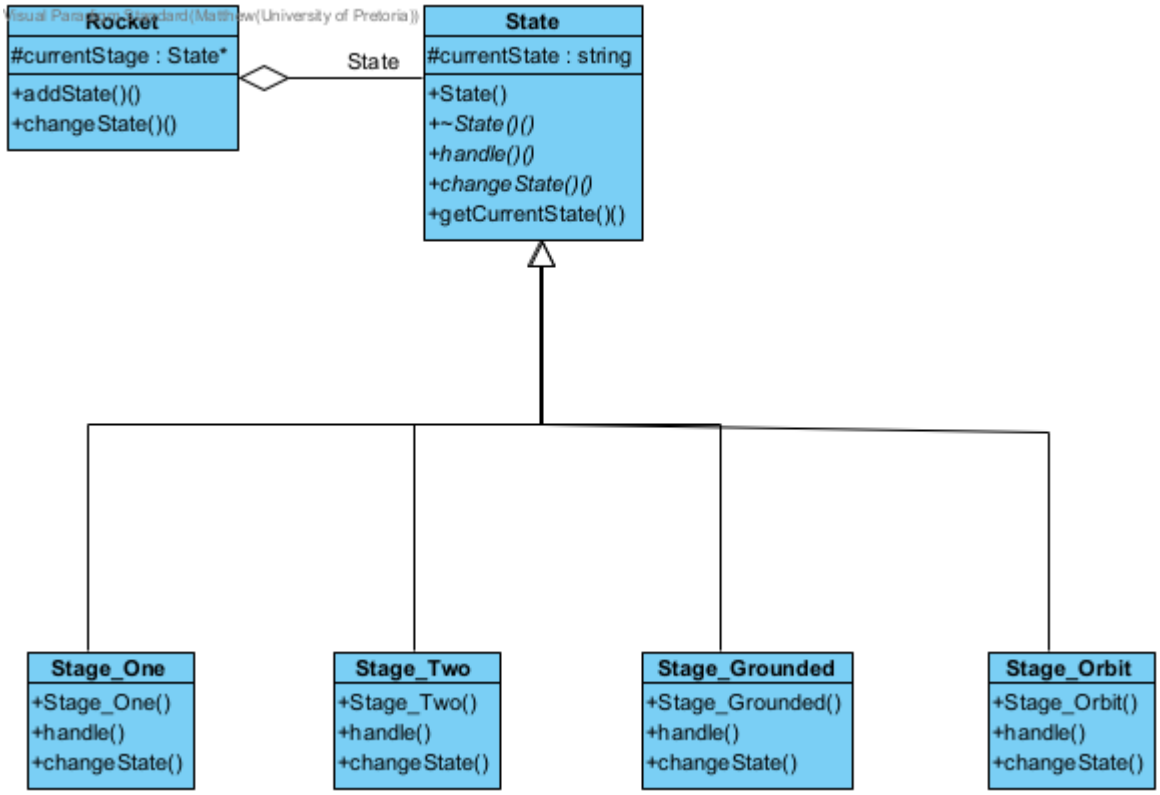
1.8)

Before Launch:	While Docked at Space Station:
	
	

Final Report:

State

https://drive.google.com/file/d/1plRu0uYPoAaoOiORhxVF3pOyHL_wgq-Q/view?usp=sharing



We used the state design pattern to keep track of the different steps of a rocket object's journey from the launchpad until it is in orbit. The state design pattern allows Rocket to change its attributes depending on what state it is in.

The Stage_Grounded state will accommodate the actions a rocket object will take before the rocket launches. The static fire test will be among these actions. This state is also a rocket object's default state

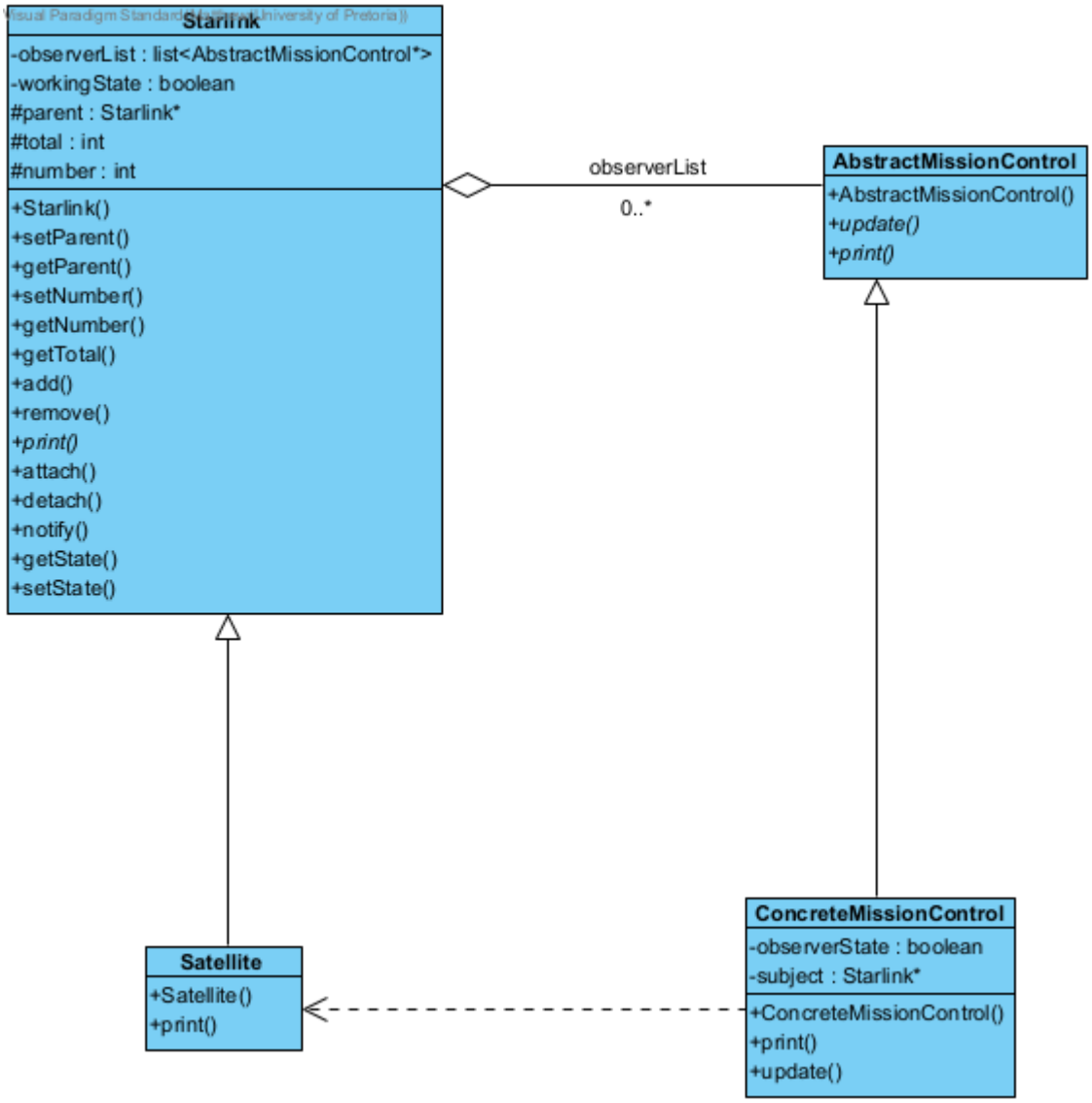
The Stage_One state is used to describe the amount of cores and engines a rocket object will have before it reaches the next part of its journey. The attributes within the rocket objects will change accordingly depending on the state.

The Stage_Two state is used to describe the amount of cores and engineers a rocket object will have before it delivers the payload to the desired orbit. The attributes within the rocket objects will change accordingly depending on the state.

The Stage_Orbit state is used to describe how the rocket object will look when it releases its payload.

Observer

https://drive.google.com/file/d/1DTatTYm9o16DyHMvKCicm-0W000uvOcE/view?usp=sharing



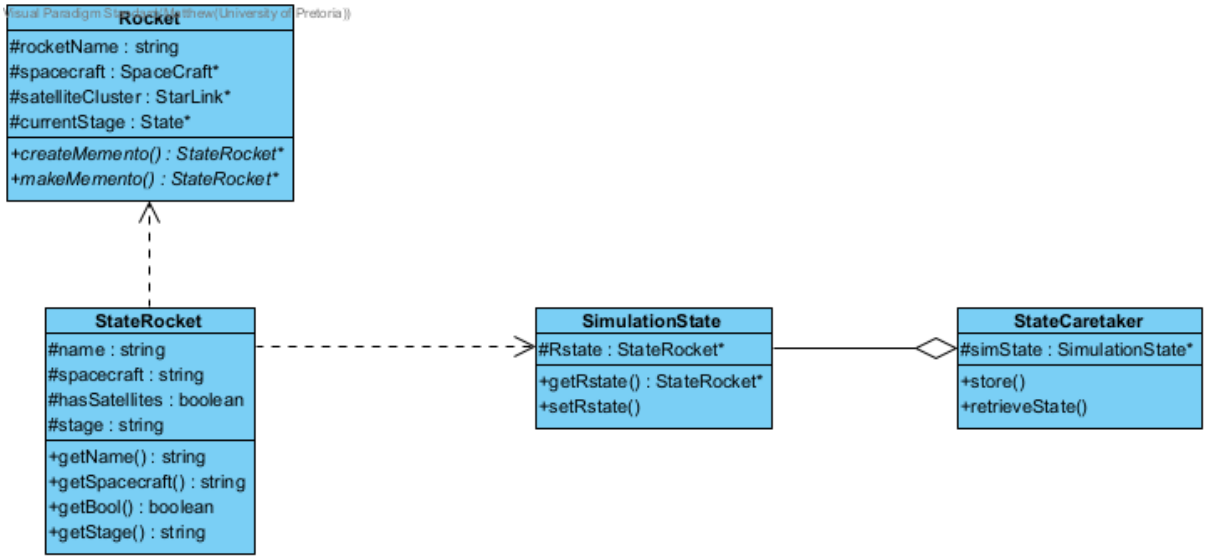
We made use of the observer design pattern because it allows for satellite objects to communicate state changes with each other, so that it can properly simulate an actual satellite network.

The AbstractMissionControl and ConcreteMissionControl classes act as the Satellite network's HQ. These classes give the satellites the update commands.

The StarLink class will communicate with all the satellites in the network when one satellite's state changes and will tell the rest of the satellites in the network to also change their state.

Memento

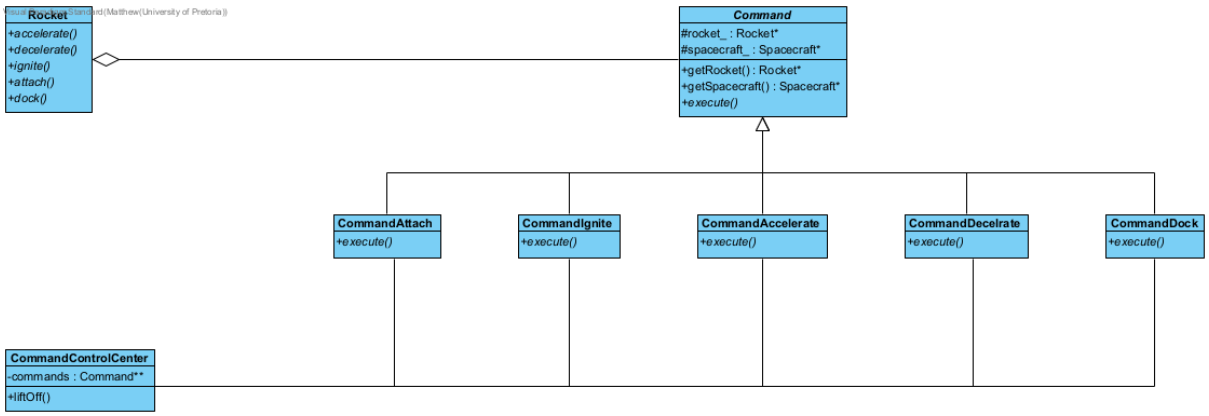
https://drive.google.com/file/d/1eB338qD8gnnbURR9bzUw5pE-K43tUMUK/view?usp=sharing



This design pattern helps us reset the simulation in case the simulation fails. The reset will change all of the Rocket Object's attributes to previous values before the simulation failure. This will prevent us from having to restart the entire simulation from the very beginning.

Command

https://drive.google.com/file/d/1ugArTw6NyrPmFLInCLYCc7Gzf5dwQOW9/view?usp=sharing

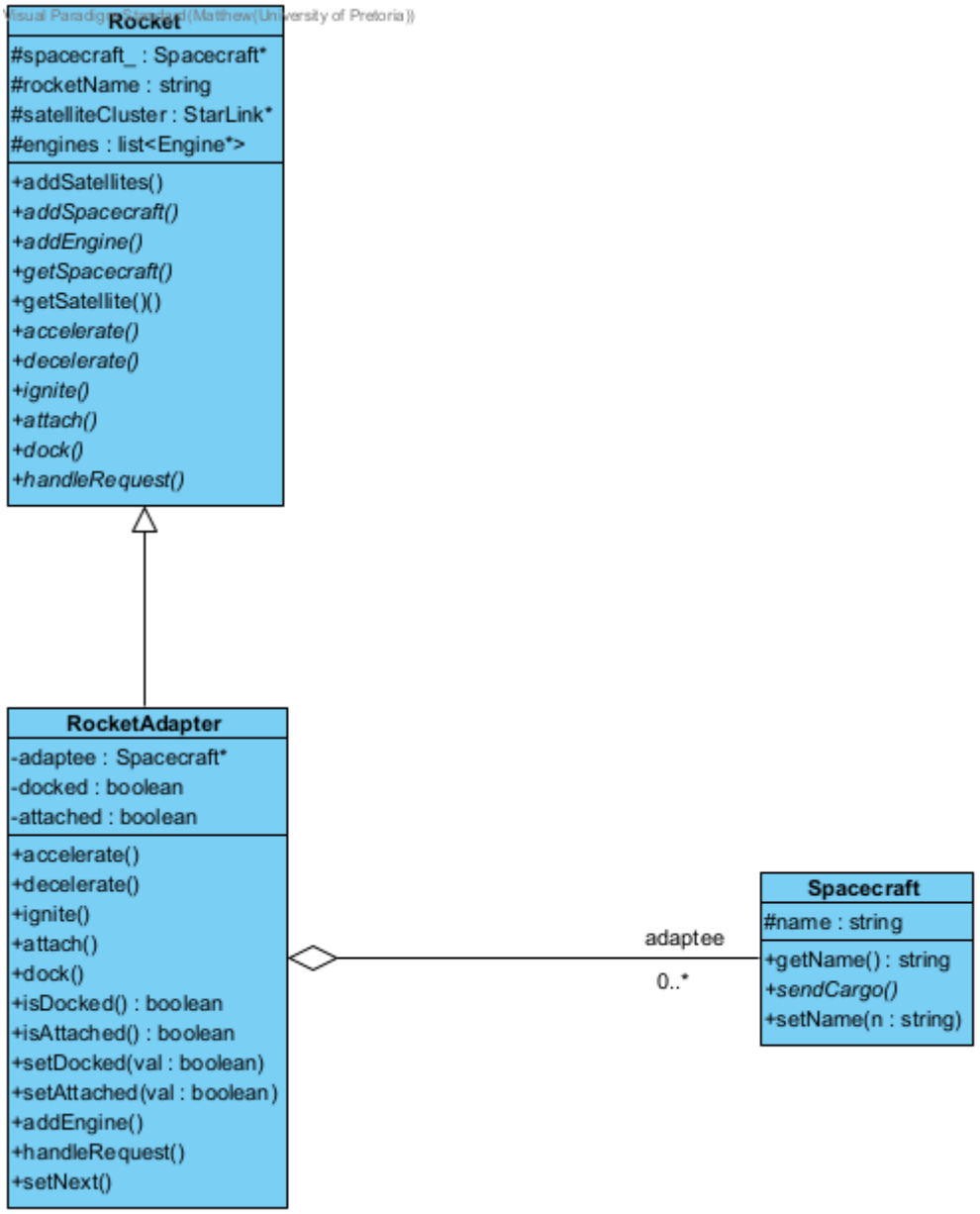


This design pattern allows us to implement an interface between the user and the Rocket object. This prevents the user from directly having to access the Rocket object's member functions. This interface will also make it easier for the user to know which actions the rocket object is allowed to perform. It also centralises control of the rocket's actions to the receiver which is CommandControlCenter. This design pattern also allows us to queue the actions the rocket should make throughout its journey so that it does not need to be done in real-time.

CommandAttach tells a spacecraft to attach itself to a rocket.
CommandIgnite is used to start the engines for the rocket.
CommandAccelerate is used to make the rocket go faster.
CommandDecelerate is used to make the rocket go slower.
CommandDock is used to dock the Spacecraft object with the space station.

Adapter

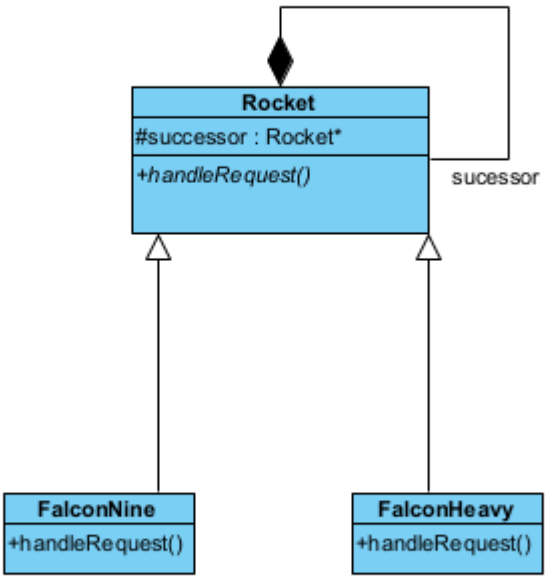
https://drive.google.com/file/d/1uGE-dgMPROqSTa3wMscJ-TNd2rgwpEa_/view?usp=sharing



This design pattern allows us to seamlessly integrate some of the Rocket object's functions with a spacecraft object. This design pattern prevents us from having to extend the Rocket and Concrete Command classes, thus preventing the need to make these classes more complicated than what they need to be.

Chain Of Responsibility

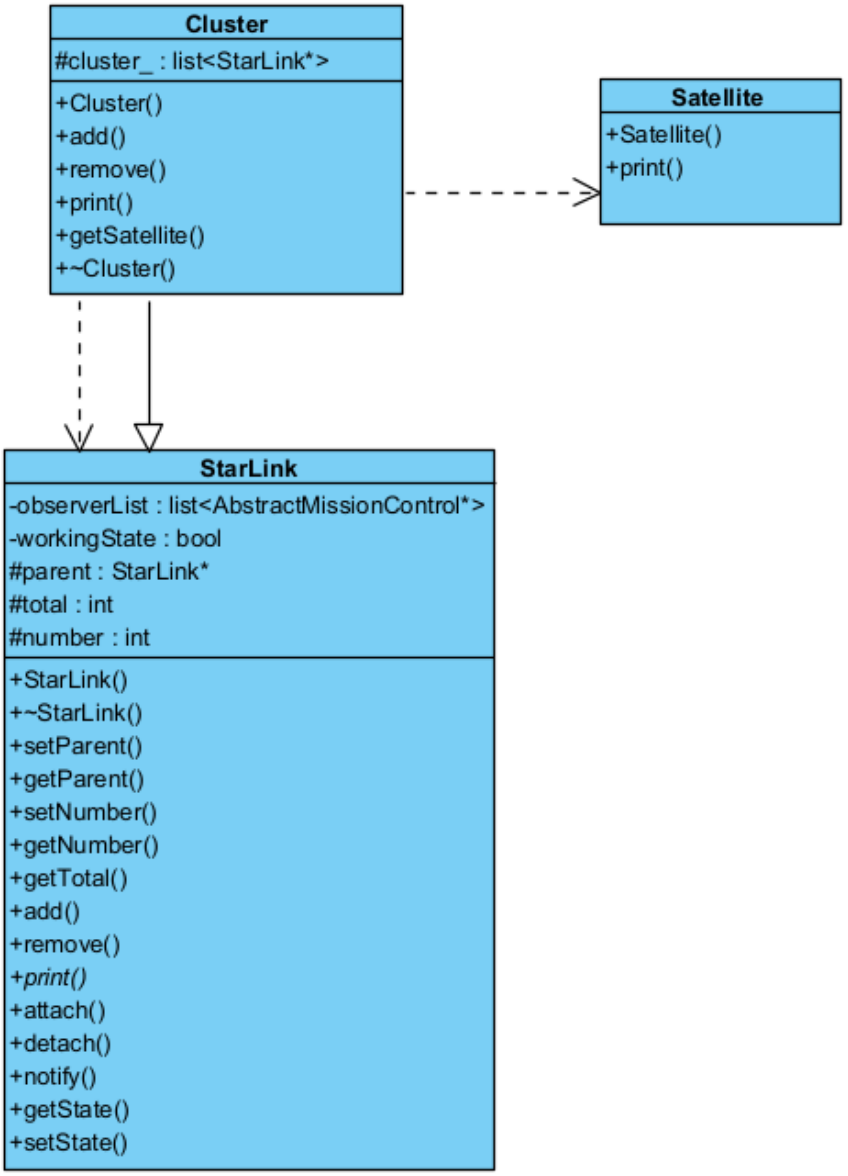
https://drive.google.com/file/d/186x0beW5wVRxKnclSPP92CnuK8Ku_QNE/view?usp=sharing



This design pattern was used because we have two variations of the Rocket object and each variation performs the same functions differently from each other. The Rocket class will determine which method process to use depending on which variation of the Rocket object called the function.

Composite

https://drive.google.com/file/d/149XLN6OlgL2_pdxfBp9sOD3XjVBAjJ_D/view?usp=sharing

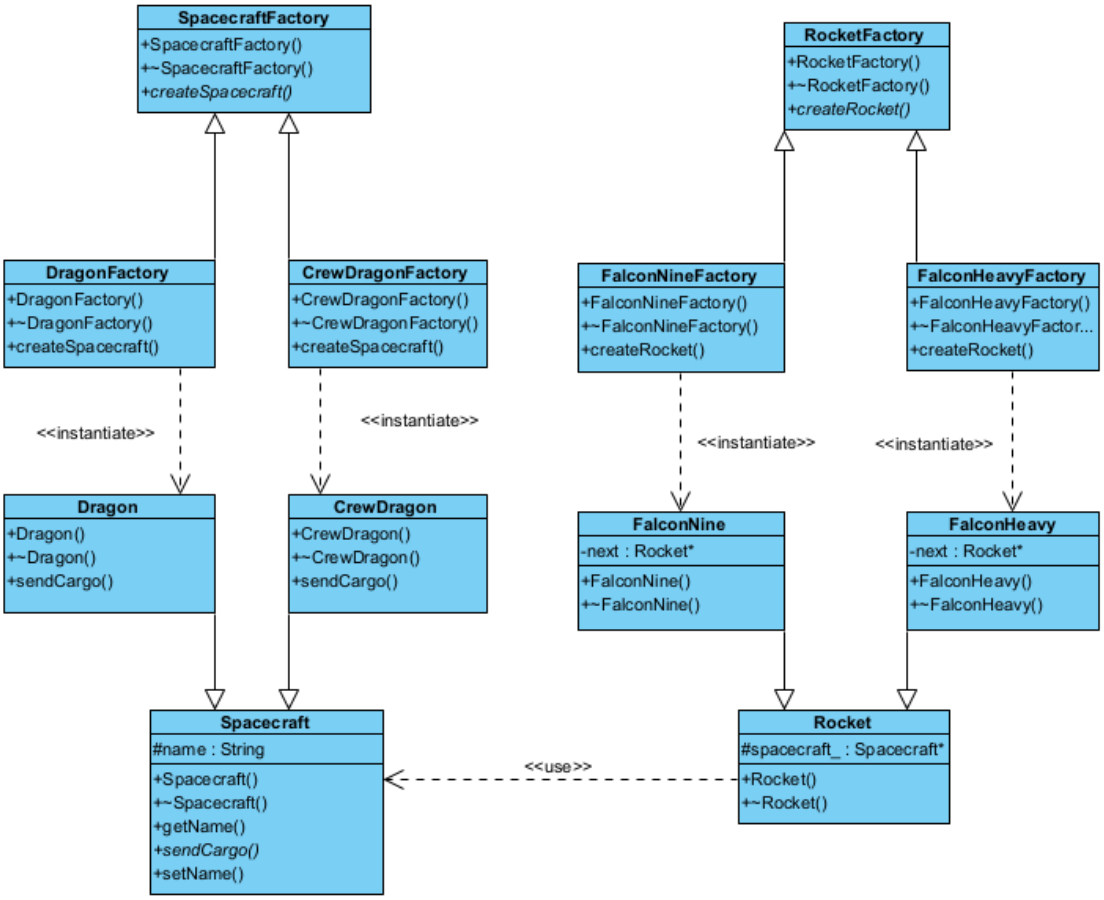


The design pattern was used because it allows for easy scalability.

The design pattern also allows us to treat each created cluster of satellites uniformly.

Factory Method

https://drive.google.com/file/d/1ZAUeQo6taGWYJklAH_LfohkjGFBVXFhL/view?usp=sharing



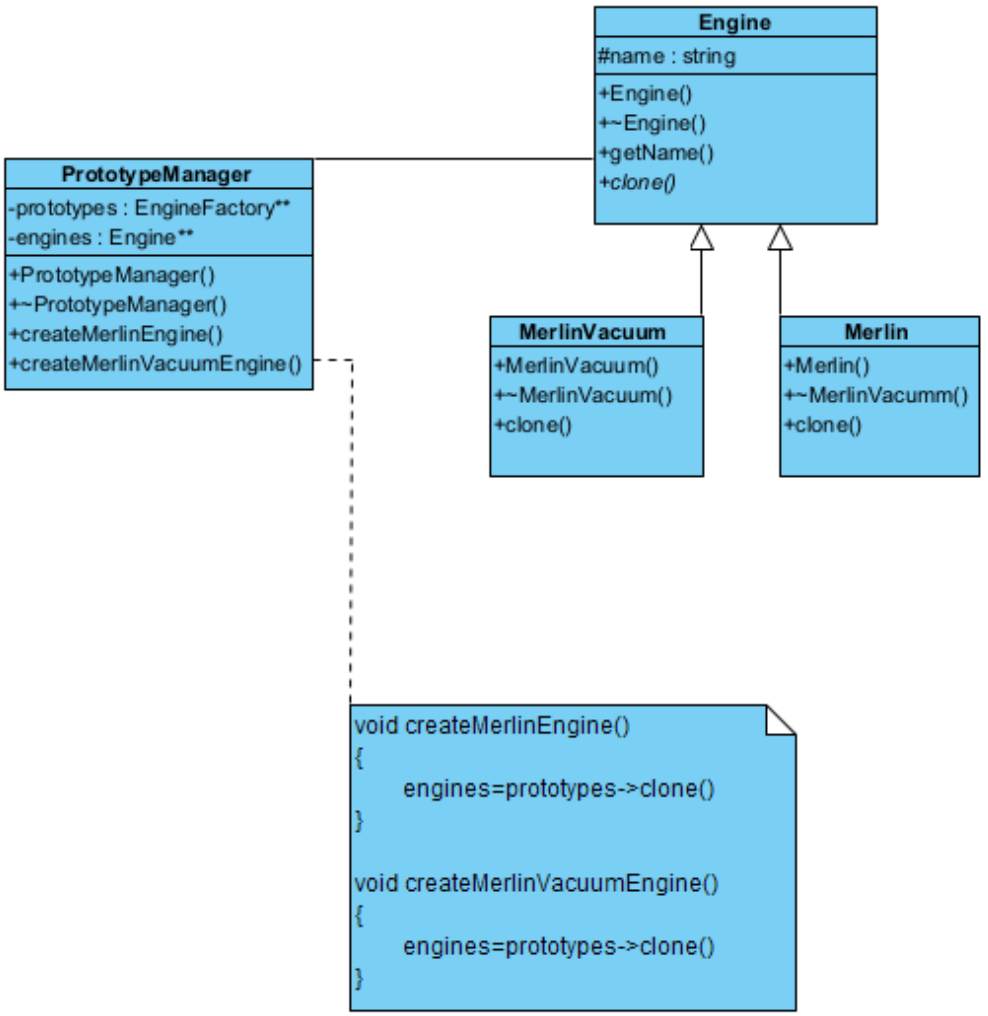
This design pattern was used because it allowed us to create specific variations of Rocket and Spacecraft objects all of which will inherit from a common source specific to that factory. This design pattern allows for scalability as concrete products and creators can just be added to the required factory.

It also centralises the creation of objects to a limited number of classes. This prevents code from being scattered around. It allows for easy amendments to be made to objects if need be, since you only need to alter the concrete creator that creates the desired concrete product.

Overarching base product classes such as Rocket and Spacecraft also make it easy to alter the structure of only the concrete products that inherit from it, allowing, once again, for easy, centralised and simple amendments to the code.

Prototype

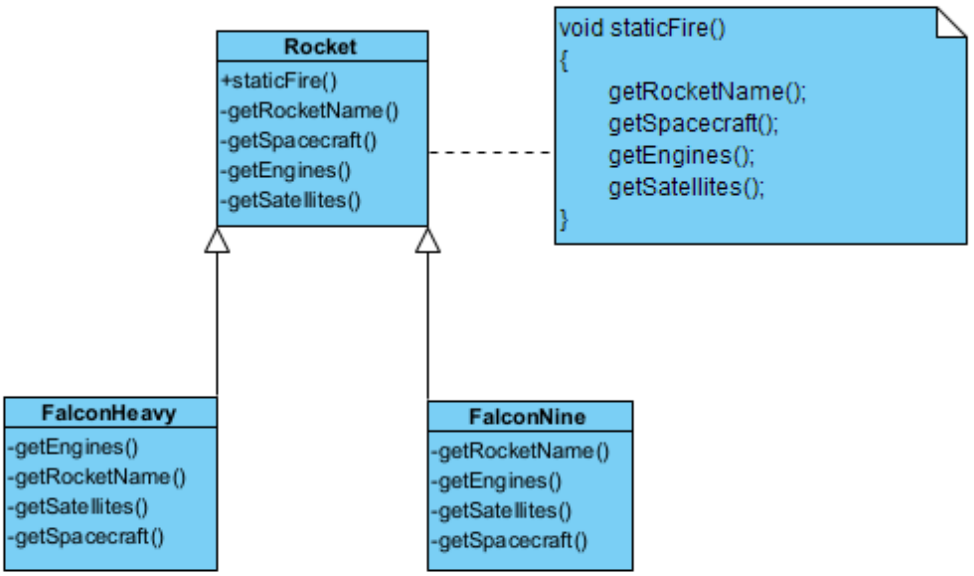
https://drive.google.com/file/d/1Uf2oGVqQziGOCkRa9A-pFbowM05a6WUJ/view?usp=sharing



Instead of inheritance being used, we use the prototype pattern to allow the client to create multiple merlin/merlin vacuum engines from a manager object. Only a single engine needs to be created and from that object, clones are created, it is useful since we can create engines with slightly different states/attributes, which we can then create clones from. The reason this may be useful is because for the simulation, you may need to alter the structure of the engines, when you find one that works, that engine will be used to create other engines upon it. Since only 1 needs to be created it is simple for the client to replicate, and with the use of the prototypeManager, it centralises that cloning process and stores them for you.

Template Method

https://drive.google.com/file/d/1TF8SNwiN5HjKLIPW41R9AhPaMauCI_F-/view?usp=sharing



This design pattern was used for the static fire test. This is because Both Falcon Nine and Falcon Heavy rockets will follow the same process, but each rocket type will perform the steps in the process differently.

The Rocket class provides the process each rocket type should follow.

Falcon Nine and Falcon Heavy define what the rocket object should do at each step of the process.