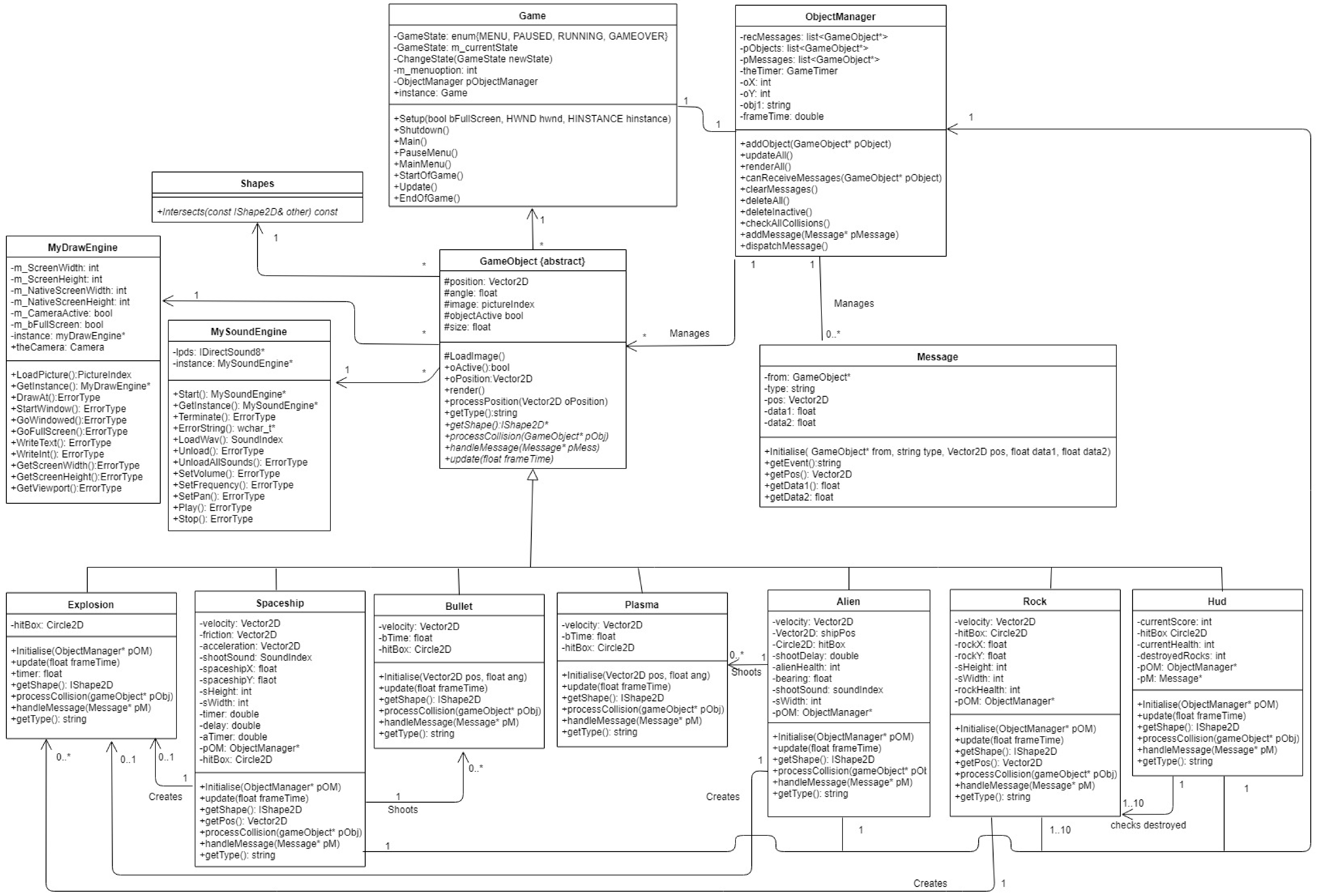
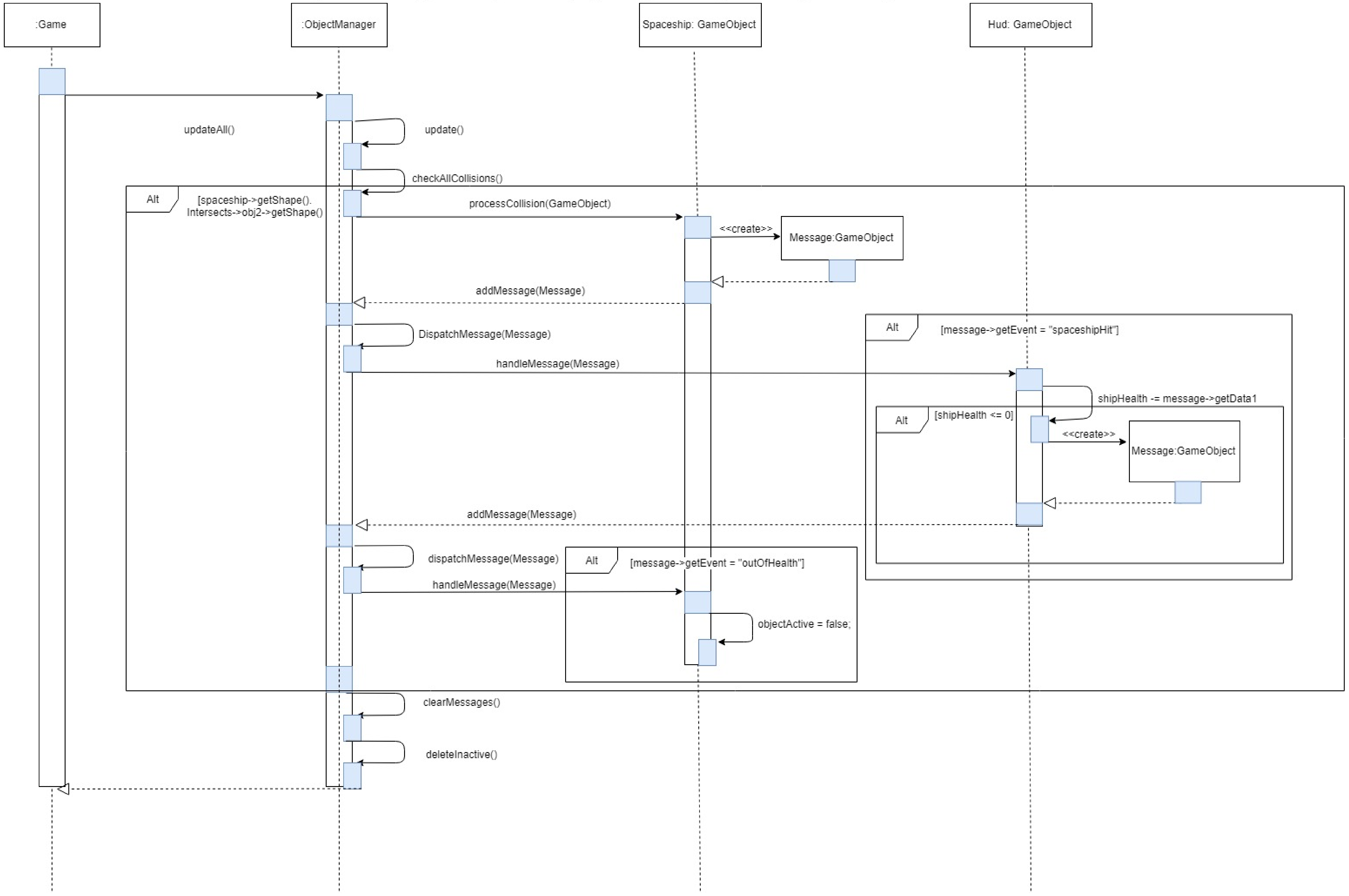
Game Engine Class Diagram



Sequence Diagram: Spaceship getting destroyed using messages



Software Architecture Essay

Student Number: W16014375

Advantages of the game engine

Game Object Class

In comparison to the API provided at the start of the assignment, this engine now makes it much easier for a programmer to create their own game in numerous ways. One way it has improved the experience for a programmer is by including an abstract GameObject class in which all game objects inherit from. This class supports the implementation of game objects easily by providing methods and attributes that all game objects should use. The GameObject class uses dynamic binding virtual functions so that game objects can overwrite functions such as processCollision or handleMessage so that the desired outcome is obtained. This also has the advantage of better organising the code so that if a programmer new to the engine, they would find it easier to understand game objects (Terence, 2017).

Object Manager Class

An ObjectManager class has been added to the engine to manage all game objects. This allows a programmer to easily add, render, update and delete objects from the game by calling the necessary function and passing the object pointer to the object manager. This is much more efficient than for example, calling update for each individual object in the game one by one.

Messaging System

The object manager also has an implemented messaging system which can send messages. Messages are objects that store information using encapsulation that has been sent from one object to all others that can receive them; game objects that can receive messages are added to a “recMessages” list when they are initialised. This allows for the dynamic creation of messages and can be easily utilised by a programmer to pass a message from one object to another, overall improving the programming experience. The message type was chosen to be a string so that it can be dynamically typed and due to the size of the engine it should not affect the engine speed too drastically as this engine is currently only suitable for smaller games and should only be used for such. This allows interaction between entities to remain manageable using this messaging system (Bevilacqua, 2015).

Cohesion

Due to implementing the GameObject and ObjectManager class, this has increased cohesion as other classes can be more focused on particular elements such as their update methods rather than having to also focus on such things as rendering because this will be handled by the game object and object manager classes. With this higher level of cohesion; it makes the code of the engine much easier to read and understand, thus making it easier to debug and test (Edaqa, 2015). This would be especially important if the engine was being worked on by more than one programmer at once.

Limitations of the game engine

Shallow Hierarchy Limitations

Due to this engine having a shallow hierarchy, this causes issues for programmers when they add new objects, because if they add an object which doesn’t use one of the pure virtual functions or attributes from GameObject then they may need to come up with a solution to bypass this. If a vertical hierarchy was used for this engine it would’ve further improved cohesion as certain classes such as VisibleObject and MovableObject could be created to allow for differences in game objects. Game objects could inherit from these classes instead of directly from the GameObject class. This would prevent the need for the programmer having to find ways to use the GameObject class if a certain game object doesn’t use all the GameObject class features.

Messaging System Limitations

The messaging system is currently limited in terms of speed as it uses stings to pass the event type. If the engine was to be built for larger games, then enums would have been chosen in favour of their higher speed in comparison to strings (Mertz, 2016). Also another issue to note with strings is that they are prone to typos that can be harder to spot than enum typos as a compiler will directly complain about unrecognised enums (Mertz, 2016).

Another issue with the messaging system, is that although the messages are only sent to objects that can process messages, these objects receive *all* the messages. For example, an alien will receive a message that a rock has been destroyed which isn't relevant to the alien object, this wastes system resources sending messages to objects that don't require them. If the engine was to become more complex it would be beneficial to alter the messaging system so that the messages have a way to be sent to a specific object rather than all objects that can receive them.

Coupling

Although the coupling of the engine has been kept to a minimum there is still a reasonable amount of coupling as a lot of classes have strong dependencies on one and other including some game objects which are dependent on other game objects. Although it is currently acceptable in its current form, if the engine was to become more complex it would create more dependencies and increase compile time (Edaqa, 2015). This could eventually lead to a large amount of errors if something was to go wrong with a class also causing errors to appear in its dependent classes. This would also mean that programmers that want to edit the engine would have to learn how more of it works to edit a single class (Hokstad, 2008).

Foreground background

Another limitation with this engine is that there is currently no way to make objects appear in the foreground and background. If a more complex game was to be made this would limit programmers as there would be no easy way to bring the player in to the foreground in which objects meant to be in the background could interfere which could create massive gameplay issues. If the game becomes more complex it would be advised to add a feature that organises where the sprites are in the foreground/background.

References

Bevilacqua, F., 2015. *How to Implement and Use a Message Queue in Your Game.* [Online]
Available at: https://gamedevelopment.tutsplus.com/tutorials/how-to-implement-and-use-a-message-queue-in-your-game--cms-25407
[Accessed 15 12 2018].

Edaqa, 2015. *Cohesion and coupling: good measures of quality.* [Online]
Available at: https://mortoray.com/2015/04/29/cohesion-and-coupling-good-measures-of-quality/
[Accessed 20 12 2018].

Hokstad, V., 2008. *Hokstad.* [Online]
Available at: https://hokstad.com/why-coupling-is-always-bad-cohesion-vs-coupling
[Accessed 20 12 2018].

Mertz, A., 2016. *Simplify c++.* [Online]
Available at: https://arne-mertz.de/2016/04/strings-vs-enumerators/
[Accessed 20 12 2018].

Terence, 2017. *Making use of inheritance in game programming.* [Online]
Available at: https://terence.terresquall.com/2017/05/making-use-of-inheritance-in-game-programming/
[Accessed 20 12 2018].