ENGS 65 Project Proposal

Dan Berman and Brett Nicholas

For our term project, we are proposing to do a remake of the classic arcade game “Asteroids”. The game would be mainly done in C++, however, we both have experience in graphics coding in Java, and so we may implement C++ code to call Java code for the graphics, rather than find and learn an open-source C++ graphics package.

This project is heavily object-oriented dependent. The game would be an overall encapsulation tracking multiple objects moving throughout the game space. There would be an object to represent the ship, plus multiple instances of objects representing both the asteroids and the “bullets” being shot at them.

Encapsulating this functionality would only require one or two levels of objects, and so we were thinking we could take it further by providing the ship its own objects to act upon: upgrades. We could have the ship be composed of multiple objects, each of which has the ability to be upgrade/changed to a different object, and that would lead to affecting the behavior of the ship. For example, there could be an object representing the ship’s engine, and when the game calls the ship’s “Fly” function to have it move, rather than have the ship calculate its movement, it would instead call the engine object’s “Move” which would tell it how it travels. Through inheritance, we could then have multiple types of Engine objects, each with the “Move” function, but each one would come up with a different result: one might have the ship accelerate/travel at a faster rate, or decelerate differently, etc. Other types of ADTs and inheritance we might implement would be the ship’s gun and the ship’s hull, so that it can shoot in different ways (and produce different bullet objects), and react differently when colliding with an asteroid.

We were also considering creating a “power-up” ADT that could spawn randomly while the game is being played, and it could provide temporary benefits to the player. Objects that would inherit the power-up ADT might include a shield that temporarily protects the ship, a weapon power-up, or a ship speed-up.

One of the biggest examples of object-oriented design would be the design of the asteroids themselves. There would be multiple types of asteroids, each one differing in their inherent properties such as equations of motion, appearance, and tendency to break upon impact. Upon colliding, some asteroids should exhibit inelastic characteristics, and others should always collide in a perfectly elastic way. Of those predisposed for elastic collisions, some might break apart into smaller pieces given a high enough collision velocity, and others should maintain their structural integrity. Some asteroids might even be a “power-up” in disguise.

What is neat about this game is that there are an infinite number of things that we could implement, if done in a true object oriented fashion. If the game is properly designed and encapsulated, we could easily add whatever features we want in their own object encapsulations, and for the most part, it shouldn’t ever affect the rest of the objects in the game. The power-up ADT would be completely independent of the ship for example; so we could easily add that at the end of the project, or not if we don’t have time. The speed-up for example wouldn’t ever directly affect the ship object/sub-object. Instead, the game would ask the ship to move, and then maybe multiply the result that the ship returns by 1.5. The fact that this can be done, or not done, and it would have no effect on the creation/functioning of the game is what makes it clear that this has to be done with heavy object oriented dependence. Otherwise, every time we added something, we’d be going and changing code throughout the program, rather than at just an upper level. Thus, the hierarchical nature of asteroids, and its need for encapsulation and abstraction, makes it a perfect exercise in proper object oriented design.