**Report on features of C++ for  
Space Invaders  
Compared to Java & Python**

By: Josh Cooper

Comparative Programming Languages – PROG1226

Instructor: Stephen Johnson

Due: Nov 24th, 2014

As a student I believe learning is just like the sky, boundless and filled with infinite space. For the learner of programming a programming language is a lot like an aircraft, the type of craft can have a huge impact on where you can go and how fast you can get there and what you consequently learn. I would argue that C++ is like a Swiss army knife capable of adapting to a vast number of situations. As this applies to game design there are a number of software design patterns employed in the creation of games. Patterns ranging from singletons to spacial partitioning and even dirty flags and double buffers. Having already familiarized myself greatly with C++ I know that C++ is more than capable of defining & describing these patterns. As such I feel C++ to be the better language for any project whose primary objective is help me as a programmer learn new concepts which will be helpful to me throughout my career. If I had to make a decision as to which language was outright best for completing the project I went with I would choose Python for its ability to accomplish programming tasks quickly. The features C++ brought to the table were numerous: Templates, Polymorphism & Inheritance, Pointers, Memory Management, References, and Random Number Distribution objects.

The project relies heavily on three management classes sharing pointers to objects. This is where templates come into the fold, because if you know you will have three classes each with a list of pointers to objects then you may be able to share code which handles the aspects surround the pointers to objects. In C++ you can define a template class which will hold an unknown type of object pointers in a container and inherit this template class into your three managing classes. If you take it a step further and create the template container, in the template class being derived from, as a singleton container then the ability to share information with these management objects becomes incredibly simple. The entire object does not even need to be defined as a singleton, just so long as the data storage is handled as a singleton. Thus we can create an instance of our managing class anywhere in our program and adding data to its container will affect all instances of this template derived class. For Java sharing objects between different containers is simple as is creating singletons, but since Java does not have templates it could be slightly difficult to replicate C++ template classes as Java Generics. As for python, you could argue that everything is already written as a template due to the lack of concrete typing.

Polymorphism comes nicely into play when creating the objects which are to be managed; allowing you to create complex objects and then manage them as base class objects with the implementations of their actual class. In Java this means creating a file for every class you are going to throw into the muck. This can slow down the implementation of these classes but not by a great deal this is more of a tedious requirement than a development problem. The Polymorphism available in python seems almost like a joke, duck typing they call it where you treat a parent object like a child object if it acts like a child. How does one explicitly treat an object like any other given object if you can’t specify how to treat it? It certainly would complicate the thought process in designing any such necessary behaviour. In C++ and Java however we can be confident that our intentions will hold firmly in place. The fact that Java, C++, and even Python have inclusion polymorphism (subtyping) is a huge plus although used but not greatly leveraged in this project.

Exception handling in C++ is only truly necessary for show stopping exceptions and not even to carry on as if nothing happened but to know that a component has become broken. With that said implementing exception handling is incredibly easy to implement in C++ and Python. In Java catching exceptions is the difficult part as you must know the exact specific exception name in order to catch it. In this project the only portions of code utilizing exception handling were components for File I/O. In C++ the File I/O was again incredibly easy to implement. In Java however, the File I/O is made confusing by the presence of multiple classes for writing and reading from files in different ways. For instance using Scanner to read a file leaves you wondering what writer you may wish to use or even need to use; this is perhaps where experience in Java would come into play, but still wonder why keep multiple classes when you could condense.

Memory management, although perhaps hard to believe, has a large stake in what appeals about programming. The ability to describe how memory is to be manipulated, accessed, stored, and even allocated. These things let you know that you are in control like no other set of tools; although this ability can sometimes lead to a misunderstanding of bugs. Debugging in this type of environment certainly requires a programmer to put to use a great number of debugging tools and approaches. Different bugs call for different approaches sometimes, but that is fine. One issue may require a logging toolset, another may need benchmarking, and yet another may require modularized code. When adding threaded execution of code into the mix things become yet even more complicated, and yet again even more so when multiple bugs are present simultaneously. Often the most useful tactic will be creating modularized code so that you can disable entire modules while narrowing the scope of a particular bug. You may find that something which could be caused by memory being thrashed has nothing to do with memory at all but rather an ill-formed assignment expression missing a `+=`. Without programming such a project in C++ one might never learn how to solve such convoluted bugs.

A large portion of ‘AAA’ games are made using OpenGL; by accomplishing this project in C++ a better familiarity has been reached with how the underlying functionality of a Triple A game is utilized. In Java this experience and familiarity would have come proxied through libGDX which utilizes OpenGL rather than working directly with OpenGL. Python would have been no different from Java except in its library pygame. As for C++ using Blit3d and the source code for Blit3d one is capable of working as closely with OpenGL as one desires.

The last and probably the most important to the project, and how it was implemented, are the random number distribution objects used in conjunction with a random number engine. In C++ there are objects to allow customized output of a pseudo random nature. What this means is if you want 3 random numbers { eg. 1, 2, 3 } to output in particular ratios you need not worry with the math of how to manipulate random numbers so that 2’s output 45% of the time whilst 3’s output 30% and 1’s at 25%; no instead all you need to concern yourself with is providing the distribution parameters to the distribution object. The random numbers similar to the above are used for deciding how many bullets are shot by the space invaders at the player. There is another distribution which outputs Boolean values which is set according to the last time the invaders shot ( threshold of ~500ms ), as to space the shots out, and how many shots are currently on screen ( max: 30 shots ). Although this could be separated into separate logic, a prelude managing when the last shots were made and if we can yet shoot again, and then have the distribution based purely off how many shots are actually on screen at one time; keeping the logic in the distribution keeps it simple enough and allows for greater difficulty to the player. There is a third far more trivial distribution which is merely from a minimum to a maximum range, which could easily be replicated in any language with random numbers. In Java & Python this is far more complicated requiring you to know more of the underlying math in order to find appropriate classes or merely implement the functionality required on your own. For the uninitiated manipulating random number generation output to fit within a particular distribution is hard to comprehend.