Iskandar Shoyusupov & Emmett Easter

Project 7 Final Report

Final State of System

Our final system is a swift app that randomly generates tiles. These tiles then fall from the top of the frame, and upon clicking them in a certain area, send the data to the raspberry pi device. The Raspberry Pi device then interprets what note to play based on the data, and plays the note through a speaker.

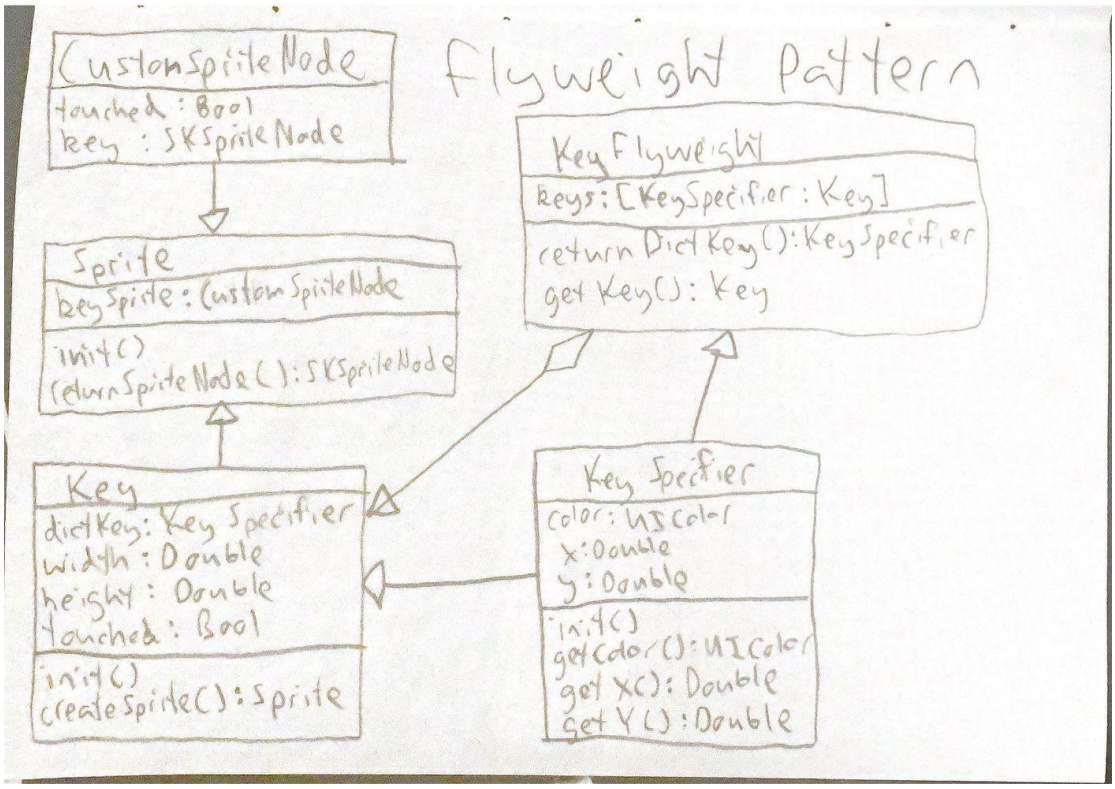
Our original system had several features that were not implemented in the final version. One of which is progressive levels based on score. The app does create multiple levels, with increasing difficulty, and keeps track of score. However, reloading the game scene after level completion with a new level was much more complicated to implement then expected. With the way game kit works, you can't reload a level inside the scene, or change variables or time. So, you have to load the game scene in a very specific way in your view controller. As a result, we didn't have time to rework the view controller to support this.

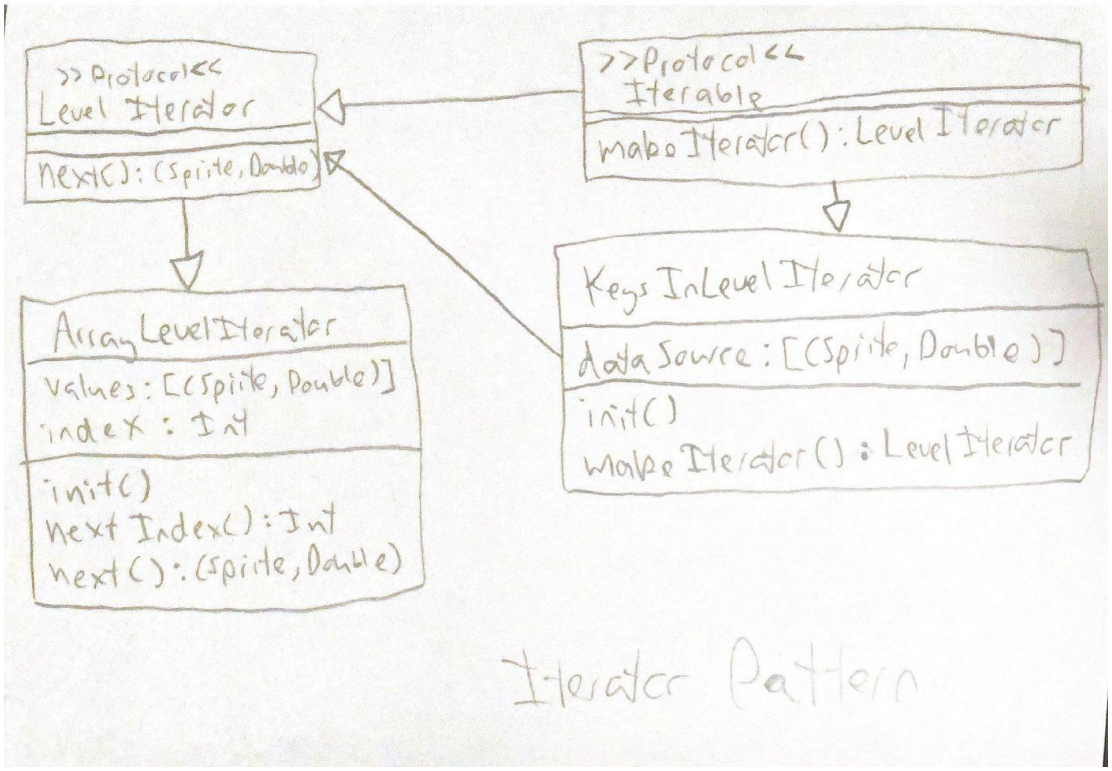
Other minor features that were not included were related to UI. The starting screen is barebones, and the game scene background is plain. It was planned that the scene background would have a solid colored stripe, indicating where the tiles had to be for you to make a sound play when clicking them. This wasn't implemented due to the scene kit having a different background system to swiftUI. We were unable to add shapes and other visual markers, so the only way to implement this background would be to create a custom image. We didn't have time to create this, so it was not added.

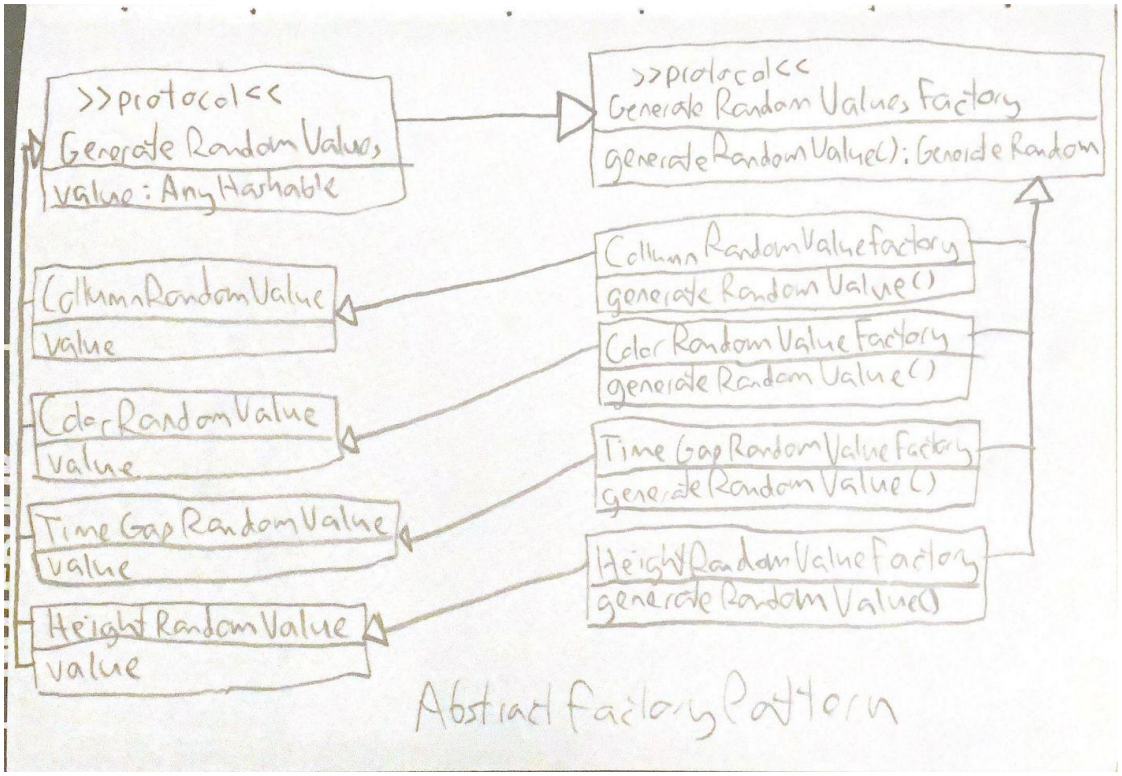
On the raspberry pi side, there was a lot of change in how the tones were being played. In the beginning the plan was to set up a tiny speaker and have the raspberry pi read in WAV file data to parse tones and play them. These tones would be stored on a HW-125 sd chip for easy access. This was not an optimal set up and brought about a lot of problems relating to data stream deciphering. Instead the model was changed to simply use the on board 3.5mm audio jack and connect it to a working speaker. Tones were not stored on the device but instead played from a python library. This made creating the application easier and let us focus on server connections, which changed from UDP to TCP for more reliable and easier to code connections.

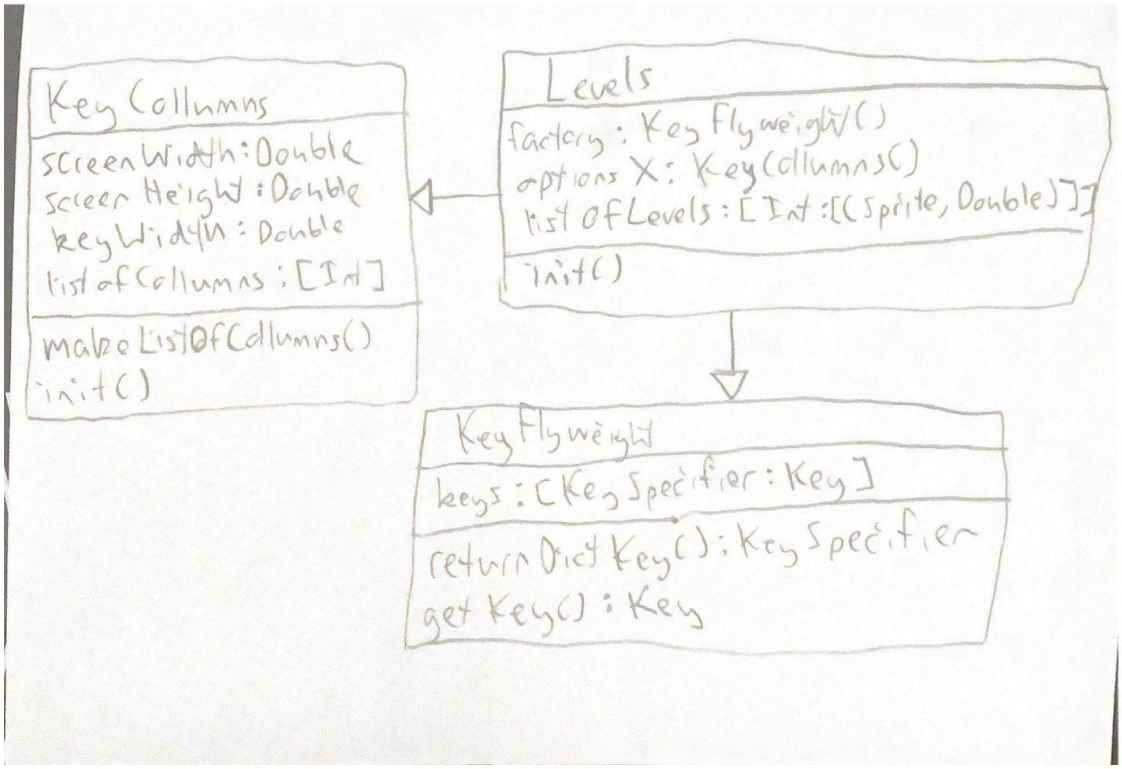
Final Class Diagram and Comparison Statement

New UML Diagram:

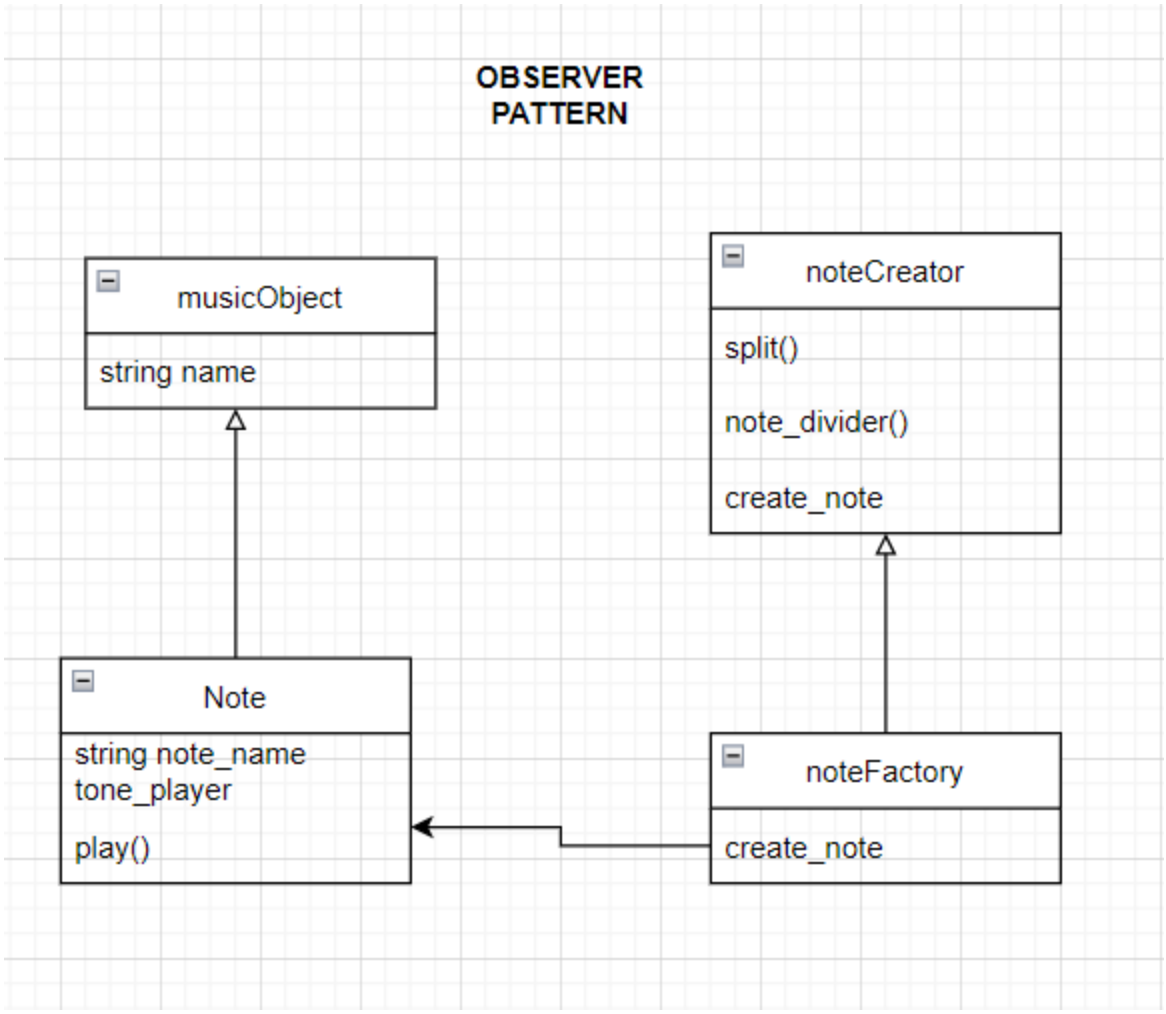




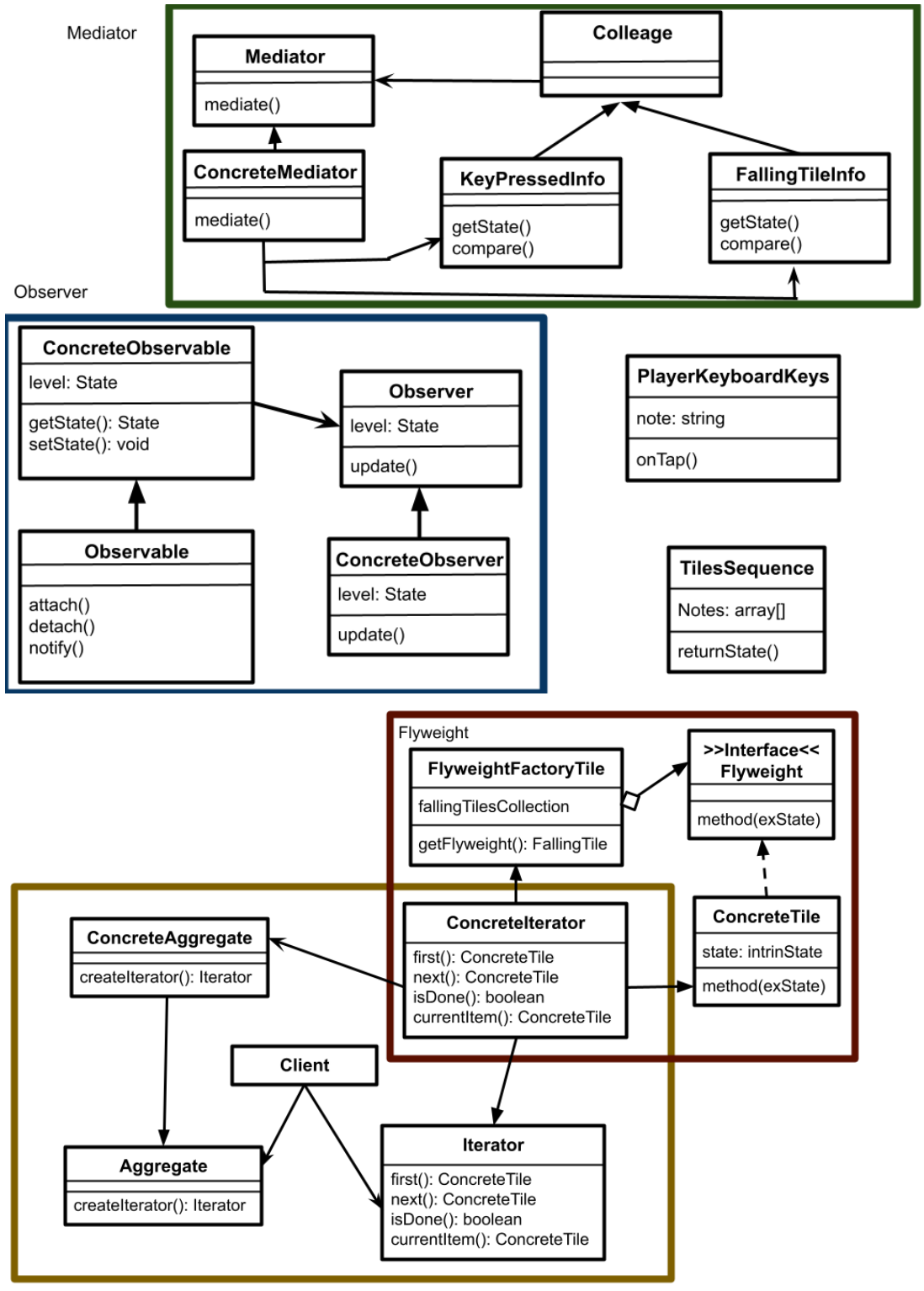

>>protocol<<
Generate Random Values
value : AnyHashable
>>protocol<<
Generate Random Values Factory
generateRandomValue(): Generate Random
ColumnRandomValue
value
ColorRandomValue
value
TimeGapRandomValue
value
HeightRandomValue
value
ColumnRandomValuefactory
generate Random Value()
Color Random Value Factory
generate Random Value()
Time Gap Random Value Factory
generate Random Value()
Height Random Value Factory
generate Random Value()
Abstract Factory Pattern


Key Columns
screen Width : Double
screen Height : Double
key Width : Double
list of Columns : [Int]
make List Of Columns()
init()
Levels
factory : Key Flyweight()
options X : Key(columns())
list of Levels : [Int : [(Sprite, Double)]]
init()
Key Flyweight
keys : [Key Specifier : Key]
return Dict Key() : Key Specifier
get Key() : Key

Iskandar Shoyusupov & Emmett Easter



Old UML Diagram:



UML key changes:

The flyweight and iterator patterns remain, although they're implementation differs
Factory pattern was added for creation of notes that would generate tones.

Third-Party code vs. Original code Statement

Source:

https://refactoring.guru/design-patterns/catalog

Use:

Basic class framework reference, specifically swift since class work was in java, and there are slight differences in syntax.

Source:

https://github.com/jsvitana/SwiftyPianoTiles/blob/master/SwiftyPianoTiles/GameViewController.swift

Use:

Referenced to learn how to load game scene

Source:

https://github.com/jsvitana/SwiftyPianoTiles/blob/master/SwiftyPianoTiles/GameScene.swift

Use:

Referenced to update game scene, and properties of SKNodes. Specifically used the sources update() and touchesBegan() function structures.

Source:

https://forums.swift.org/t/how-to-convert-that-python-code-code-on-swift-socket-client-server/46932

Use:

Example for how to send data between machines through server and client connection via SwiftNIO

Source:
https://stackoverflow.com/questions/73925119/swiftnio-tcp-server-not-sending-data-back

Use:
Debugging problem when setting up SwiftNIO

Source:
https://wiki.python.org/moin/TcpCommunication

Use:
Basics of how to create a TCP server in python

https://wiki.python.org/moin/TcpCommunication

OOAD process for the overall Semester Project

1: Designing UML/classes

Throughout the project we had to develop UML and class diagrams to flesh out our design patterns and overall implementation of the project. It was especially interesting to work on since this wasn't a purely software based project, so some of the designs had to incorporate a hardware component as well. Overall UML design was a positive but sometimes difficult process that helped us make this project easier to manage and understand

2: Flexibility

Flexibility was a major part of the project especially on the application side. Given that multiple different devices could be able to play the app, the size of tiles, overall division of the game background, and more had to change with the size of the device. We also had to be flexible in designing the note playing algorithms because we didn't want the same notes to play no matter the level. Even though changing levels wasn't quite implemented, through the factory pattern we gave the ability for different sounding notes on each level.

3: Requirements

Requirements are part of any project and they really define the scope of what we are doing. It's always easy to create requirements in the beginning and it sometimes creates the

possibility of overestimating how much the team can do. In our case, we made requirements for a fairly polished game that would have multiple levels, and on a large part we were able to meet those requirements, from the server communication, the varying levels. The only requirements that we didn't quite reach was scene loading for the different levels and some correlation, and that mostly came from implementations being harder than we assumed before. So for future projects we have learned that before creating requirements and a general outlook for what will be part of the project, a lot of research has to be done to really determine how hard something will be to implement.