1.1 Use Case Exceptions

Test Case ID: User_Input_1
Author: Joe Skimmons, jws2191
Summary: Verify that input into the system is valid and is either a move or the action ‘z’ to quit the game.
Precondition: There is an existing array containing all valid input, and input can be taken into the system.

Test Data	Expected Result	Actual Result	Pass/Fail
r,p,s,l,k	Valid input, game proceeds	Valid input, game proceeds	Pass
‘Phrase’, ‘$%^&’	Invalid input, game stops for more input	Invalid input, game stops for more input	Pass
‘R,P,S,L,K’	Valid Input, game proceeds	Invalid Input, game waits for more input	Fail
z	Game stops	Game stops	Pass

Postcondition: Input is valid and is used to play the game against the computer

1.2 Class relationships

```
//Passing children to the parent
Public Employee(Address a, NameInfo n){};
//Passing parent to the children
Public NameInfo(String name, Employee parent){};
```

1.3 Identity, Behavior, State

To determine if a class was a “manager”, I would look for a class that has more “intelligence” or knowledge than other classes. This class would arrange for things to happen, but not perform any actions itself. The fields may contain instances of other classes. These classes may have a javadoc that describes decisions that are made for other classes, or code that supports that idea.

To determine if a class was a “helper” I would look for smaller, “dumber” classes that only perform singular tasks or responsibilities in order to assist the larger mechanism. I would look for words like “performs” in the javadoc, and a singular purpose or responsibility in the code, probably carried out by methods.

To determine if a class was a “library” I would look for a class that contains information that is used by other classes to perform their responsibilities. In the javadoc, it would probably describe a class that holds information or keeps track of things needed by all other classes to do their jobs.

1.4 Identifying Responsibilities

The boundary objects would be the Talker class and the Tester class because they interact with the user and take input. You would look for a class that is not very trusting of the user, and one that uses methods to pull sensitive and important information and does not give the user direct access. For this reason, you would see many try-catch scenarios and exceptions coded because they handle malicious or ignorant input from the user and make it compatible with the project and design. You would hope not to find a system that “trusts” the user and does not have checks for what information is coming in and how it will affect the system.

1.5 Identifying Neighborhoods

I would define a neighborhood as being a set between 3 and 5 classes where a majority of them are related in some way, or over 50%. You could examine if most of the classes are connected to a common class.

1.6 Names

There are two prominent types of Hungarian Notation. Apps Hungarian was the original and Systems Hungarian is the more recent version. They were developed in order to convey more information about the variables when deciding on names. This is not as useful in java due to its “strongly typed, declarative, and open” nature. Because java is “strongly typed”, the type of the variable cannot change unless reassigned, so naming a variable intx vs x would not matter because the compiler will notice any errors in assignment anyway.

1.7 Fields

One public field is the public static int MAX_VALUE in java.intger.
This is protected by being final, and a constant;.
One public static field is the public static int MIN_VALUE in java.integer.
This is protected by being final, and a constant.