Sclack

SOEN6411: A Roguelike in Scala
Presented to Dr. Constantinides

Simon Symeonidis
5887887

November 30, 2013

Contents

List of Figures

Listings

::Section 1::

Summary

This is my coursework for SOEN6411, on the *Scala* programming language. Inside this document we discus the different powers of the programming language, the flexibilities, and other shortcommings.

1.1 The name 'Sclack'

I personally find roguelikes a good practice for any aspiring programmer. The reason is because the ideas of such games are not too complicated to understand, but the implementation can end up being very tricky, and very messy if not planned beforehand. Apart from the design and implementation aspects, one can add their creative input as well to give a new spin to these games, with the least effort devoted to graphics programming.

I've written another application in the past called *slack* as it was another roguelike (and more of a test of data driven systems), combining the words 'slash' and 'hack'. Adding *Scala* to the equation, we finally get *sclack*.

1.2 Resources

I personally, mainly used a tutorial for getting used to the language, which demonstrates the differences and similarities of Java as opposed to Scala [1]. The Scala online API was also used numerous times to get some things working properly on the application [2].

::Section 2::

Language Exploration

We explore the language in this section and demonstrate some of its interesting features.

2.1 Classes, Objects

We are given the possibility to specify things we want to instantiate, as well as things we only wish to use statically. For instantiations, we define classes, by using the keyword *class*. For static use (for example if we have a 'Builder' class, and wish to generate objects, given a class) we use the keyword *object*. Listing 1 demonstrates a class definition. Listing 2 demonstrates an object definition. The main entry point of this toy application is shown in Listing 3.

```
1  class Person(name : String) {
2    var age  : Int = 18
3
4    def greet(name : String) : String =
5      "Hello " + name + ", my name is " + this.name
6
7    def think : Unit = {
8      print("thinking ")
9      print("doing ")
10     println("sleeping")
11   }
12 }
```

Listing 1: A Class Definition

Constructor methods are 'factored' out by adding the initialization parameters on the right side of the class name. For example specifying a name for the class Person in listing 1.

```
1  object ArrayBuilder {
2    def makeStrings : Array[String] =
3      Array.fill(10){ "Hey Listen!" }
4
5    def makeInts : Array[Int] =
6      Array.fill(10){ 12 }
7
8    def makeBools : Array[Boolean] =
9      Array.fill(10){ math.random.round == 1 }
10 }
```

Listing 2: An Object Definition

```
1  object Main {
2    def main(args : Array[String]) {
3      var person = new Person("jon")
4      var arrbool = ArrayBuilder.makeBools
5      println(person.greet("simon"))
6      person.think
7    }
8  }
```

Listing 3: Main entry

2.2 Traits ≠ Interfaces

In Scala, we are given the possibility to use Traits. Traits are like interfaces in the Java programming language, however they provide the additional possibility of introducing behavior to classese or object. We can use them by appending the *with* keyword. They may be thought as mixins, similar to those of modules in *Ruby*.

```
1 trait Observable {
2   /* requires an implementation that returns string */
3   def observe : String
4
5   /* Mixin */
6   def observeThoroughly : String = {
7     "Stop looking at me! " +
8     "Seriously that's creeping me out!" +
9     "Oh my god!!"
10  }
11 }
```

Listing 4: A Trait Example

```
1 class Human {
2   var age : Int = 18
3 }
```

Listing 5: Human to Inherit

```
1 class Person extends Human with Observable {
2   def observe : String = "A fine fellow"
3 }
```

Listing 6: A Defined Person

```
1 object Main {
2   def main(args : Array[String]) {
3     var person = new Person
4     println(person.observe)
5     println(person.observeThoroughly)
6   }
7 }
8 /*
9   A fine fellow
10  Stop looking at me! Seriously that's creeping me out!Oh my god!!
11 */
```

Listing 7: Testing out the trait

2.3 Types

We go over some of the very basic types that are provided in Scala. Not many surprises are in store here. The way you handle types is identical in the way that they are used in Java, with slightly different syntax.

Unit is the only quite different addition to the *Scala* programming language in respect to Java. Unit can be thought as a 'void' type. This is particularly useful when we want to enforce a method to return nothing upon termination of execution. The reason we might wish this possibility is because the last statement evaluated in a code block, is what is returned.

Lists are usually expressed as arrays. We are given some useful operations that aid us in managing lists, such as *reverse, collect, filter, flatten, head, tail* and more. We are given the possibility to use the infix operators ++, :+, :\, :/, etc. that create new lists instead of modifying the current ones in memory - similar to common lisp. There exist facilities to modify the in memory structures as well.

Strings are also treated as lists. As such, it is possible to invoke methods *head* and *tail* to the string "Hello World", and return 'h' and 'ello world' respectively. It is also possible to invoke other common method functions such as *map* and *filter* in order to perform more flexible operations. For example *"hello world".filter(c =⟩ c == 'h')* would return 'hh', and *"abc".map(c =⟩ c.toInt)* would return an array of the integer values of the characters.

Tuples exist in scala, and the way to declare them is by ensnaring the different elements in parentheses. For example, the tuple *(12, 32, 13.0, "Agamemnon", true)* is a valid tuple.

2.4 Pattern Matching

We are able to do some extensive pattern matching according to [3]. We demonstrate this with a tamed example in Listing 8, and with more power in Listing 9.

```
1  object Tamed {
2    def main(args : Array[String]) {
3      println(testmatch("hello"))
4      println(testmatch("how are you?"))
5      println(testmatch("MEOOOWWW"))
6    }
7
8    def testmatch(str : String) : String =
9      str match {
10       case "hello" => "hi there!"
11       case "how are you?" => "pretty good yourself?"
12       case _ => "I did not understand that."
13     }
14 }
15
16 /* Output:
17
18   hi there!
19   pretty good yourself?
20   I did not understand that.
21
22 */
```

Listing 8: 'Tamed' approach in Pattern Matching

```
1
2 object Powerful {
3   def main(arg : Array[String]) {
```

```
4     println(testmatch(12))
5     println(testmatch(12.12))
6     println(testmatch(true))
7     println(testmatch("yes"))
8   }
9
10  def testmatch(arg : Any) : String =
11  arg match {
12    case _:Int     => "You supplied an integer!"
13    case _:Double  => "You supplied a double!"
14    case _:Boolean => "FALSE!"
15    case _:String  => "Strings? BO-RING."
16  }
17 }
18
19 /* Output:
20
21   You supplied an integer!
22   You supplied a double!
23   FALSE!
24   Strings? BO-RING.
25
26 */
```

Listing 9: Powerful approach in Pattern Matching

2.5 Recursion

It is possible to perform recursion in scala. Listing 10 shows us such capabilities.

```
1 object Factorial {
2   def main(args : Array[String]) {
3     println(factorial(10))
4   }
5
6   def factorial(n : Int) : Int = {
7     if (n == 0) return 1
8     else        return n * factorial(n - 1)
9   }
10 }
```

Listing 10: Factorial in Scala

2.6 Anonymous Variables, and Lambdas

We are given the possibility to use anonymous variables and lambdas in Scala. Anonymous variables use the underscore character similarly to other programming languages such as *Prolog*, and *Erlang*. Listing 9 shows us such use. Another possibility to use these variables arises when we're using functions that require lambdas. For example, we can see this in Listing 11. You can also notice how the underscore wildcard is used in order to create a generic function for printing any element from any array.

```
1 object LambdaExample {
2   def main(args : Array[String]) {
3     var arr = Array.fill(10){ 0 }
4     var arrstr = Array.fill(10){ "Mmm!?" }
```

```
 5     printArr(arr)
 6     arr = arr.map{ _ + 1 }
 7     printArr(arr)
 8     arr = arr.map{ math.random.round.toInt * 2 * _ }
 9     printArr(arr)
10     arr = arr.filter{ _ == 2 }
11     printArr(arr)
12     printArr(arrstr)
13   }
14
15   def printArr(arr : Array[_]) : Unit = {
16     for(el <- arr)
17       print(el + " ")
18     println
19   }
20 }
21
22 /* Output :
23 0 0 0 0 0 0 0 0 0 0
24 1 1 1 1 1 1 1 1 1 1
25 2 0 0 0 0 2 0 2 0 2
26 2 2 2 2
27 Mum!? Mum!? Mum!? Mum!? Mum!? Mum!? Mum!? Mum!? Mum!? Mum!?
28 */
```

Listing 11: Lambda And Anonymous Variables

::Section 3::

Dissecting the Problem

We decompose different aspects of the given problem into smaller parts, and see how the language can help us tackle the given problems effectively.

3.1 Knowledge Representation

One of the things that you have to take care of in roguelikes, is representing different aspects of the game in the computer. You make a programming metaphor out of a board game usually. That is, you need to program a grid, which can contain different properties of the space, and add other structures to store state.

I distinctively noted *properties* because though it might be favorable to represent a room with a single, two dimensional array, we need to provide more usability. For example if we had a cell being occupied with the number '1' and we wanted to use that as a wall that would be fine. But what if it was a wall with a scribbled message on it?

This calls for a different way of structuring the map. There's two possible ways: one is to use layers, and the other is to use tiles. The layered approach follows a very simple but effictive strategy in order to tackle the problem. As the name implies, there are different layers of arrays, each of them holding a specific value for a particular coordinate. Therefore we can express each coordinate with different properties. The other way is defining a 'Tile' type, which could handle operations as such (observing the tile, looking for items, speaking to an npc etc). I wanted to stick to something simple for this application, since the aim was not to complete a full blown game, but to try out the language.

3.2 Controls

Since we decided to go for a more graphical approach to the game, we can use interface features in order to help the user navigate around the world. For this game we'll have the directional buttons *(up, left, right, down)*,

3.3 World Visualization

Usually roguelikes are represented by ascii characters on a terminal. Since we will be using a user interface library (Scala Swing, with some raw imports from java to make everything work, namely some awt parts), we will be representing the world with tilesets.

Tileset : is an image file that contains smaller images that can be used repeatedly as a pattern. These smaller segments are called tiles. Tiles usually have a set width and height as their name implies. These dimensions are set once, and are used to 'cut' out the rest (since they split the image like a grid).

We assign each tile an id. We create an Array with ids demonstrating the respective tile. The array is a 2d array, and each cell's index (x, y) is used as the map's index. Finally using these simple configurations, we are able to iterate throughout the whole map programmatically, and later on use the ids in order to draw the respective tiles on the WorldWidget frame, by overriding the paint method. The array creation for this project can be found in listing 58.

The implementation of this can be seen in the Map Factory, in Listing 58, and in Listing 23.

::Section 4::

Implementation

This part describes how to install the building tools of the application, and build the project. Some more background infor is given about these tools, and their advantages in using them.

4.1 SBT: Simple Build Tool

I like using command line tools. So, in this application, I used a build tool called sbt, and vim for writing up the code. The setup is minimal.

SBT can be accessed in the command line after you have installed it on your system. Listing 12 shows how to install the tool on different Linux variants. Listing 13 is the actual SBT file.

```
# Archlinux
$ pacman -S sbt

# Fedora, and Scientific Linux (and other similar red-hat variants) use yum:
$ sudo yum install sbt

# Ubuntu and similar use apt-get
$ sudo apt-get install sbt
```

Listing 12: Installing SBT on different Linux Variants

```
import AssemblyKeys._

assemblySettings

name         := "sclack"

version      := "1.0"

scalaVersion := "2.10.0"

libraryDependencies ++= Seq(
  "org.scalatest"  % "scalatest_2.10"      % "1.9.1" % "test",
  "org.scalaquery" % "scalaquery_2.9.0-1" % "0.9.5",
  "org.xerial"     % "sqlite-jdbc"         % "3.6.16",
  "org.scala-lang" % "scala-swing"         % "2.10.0"
)

exportJars := true
```

Listing 13: SBT File

For this project, I decided using a few 3rd party libraries which some I did not completely have the chance to investigate given the alotted time. You can see these libraries inside "libraryDependencies". The first column is the GroupID (think of it as a certain software vendor id) we want things from, the second the library name, with a required scala version shown some times, and the third column being the version of that actual library.

For the purpose of this project, I added Scalatest, SQLite, ScalaSwing and ScalaQuery to the dependency stack, though as stated, I did not get a chance to use everything. I've also used a plugin to sbt called *Assembly* which packages all the dependencies in a big jar file, so that you can share the resulting binary with ease. (This however will make a really fat jar file, and might not be something you want, so be sure that you want to indeed assemble things).

Note: an SBT configuration file (project-name.sbt), IS A scala file. So you can probably do some programmatic stuff in there; I have not tested this out yet. If you want to read more about this file, you can read the official documentation here:

- SBT File Specification: `http://goo.gl/klcF93`
- Detailed SBT File Specification: `http://goo.gl/M7reZa`

You can run the tool by typing in *sbt* in the command line. On a last note, as previously mentioned, the sbt file is not really required for a project. You can enter any directory with Scala sources, and run sbt in 'project-root-dir', you're ready to compile and run your application.

4.1.1 SBT Commands

The only commands you really need to memorize, are 'compile' and 'run'. And 'run' calls 'compile' if needed. So that's one command.

If you're using a testing framework as I am (scalatest), you can type 'test' in order to run your tests.

If 'scaladoc' is available on your scala install, you should be able to run the command 'doc' in sbt, and that will generate API documentation given that you have written javadoc like comments.

```
1 [psyomn@aeolus sclack 0]$ sbt
2 Loading /usr/share/java/sbt/bin/sbt-launch-lib.bash
3 [info] Loading project definition from .../scala/sclack/project
4 [info] Set current project to sclack (in build file:.../scala/sclack/)
5 > test
6 [info] WizardTest:
7 [info] Wizard traits
8 [info] - should be an instance of Character
9 [info] - should be observable
10 [info] - should be demonstratable
11 [info] Wizard attributes
12 [info] TileHelperTest:
13 [info] TileHelper
14 [info]   Attributes
15 [info]   - should include a tile method
16 [info]   - should include a width method
17 [info]   - should include a height method
18 [info]   - should include a tileset method
19 ...
```

Listing 14: Scala Test

::Section 5::

Code

This section lists all the code that was used in order to write up *Sclack*.

5.1 User Interface Implementation

```
1  package sclack.ui
2
3  import javax.swing.ImageIcon
4  import java.awt.Dimension
5  import swing._
6  import swing.event._
7  import scala.swing._
8  import scala.collection.mutable.ListBuffer
9
10 import sclack.domain.commands.Commandable
11
12 /**
13  * Generic menu to choose a game to load or delete (those UIs should delegate
14  * or inherit from this class).
15  *
16  * @see     DeleteGame
17  * @see     LoadGame
18  * @author Simon Symeonidis
19  * @note thanks to http://stackoverflow.com/questions/6809305 for listbuffer
20  *   and listview usage
21  */
22 class GenericGameSelector(actionName: String, command: Commandable) extends
      Dialog {
23
24   val chooseButton  = new Button{text = actionName}
25
26   val gameListLabel = new Label{text = "Games:"}
27
28   val gameList      = new ListBuffer[String]()
29     gameList.appendAll(List("Savegame 1", "Savegame 2", "Savegame 3"))
30
31   val gameListView  = new ListView[String](gameList)
32
33   var components : Array[Component] =
34     Array(gameListLabel, gameListView, chooseButton)
35
36   title = "Choose game to " + actionName
37   modal = true
38
39   /* Widget Dimensions */
40   preferredSize = new Dimension(300,300)
41   maximumSize   = new Dimension(300,300)
42   minimumSize   = new Dimension(300,300)
43
44   contents = new BoxPanel(Orientation.Vertical) {
45     contents ++= components
46   }
47
48   listenTo(chooseButton)
49   reactions += {
50     case ButtonClicked(b) =>
51       b.text match {
```

```
52          case `actionName` => command.execute
53        }
54    }
55
56    centerOnScreen()
57    open()
58  }
```

Listing 15: Generic Game Selector

```
1  package sclack.ui
2
3  import javax.swing.ImageIcon
4  import java.awt.Dimension
5  import swing._
6  import swing.event._
7  import scala.swing._
8
9  /**
10  * Menu to load a previously saved game.
11  *
12  * @see     DeleteGame
13  * @see     LoadGame
14  * @author Simon Symeonidis
15  */
16 object LoadGame {
17
18 }
```

Listing 16: Load Game

```
1  package sclack.ui
2
3  import javax.swing.ImageIcon
4  import javax.swing.border.EmptyBorder
5  import java.awt.Dimension
6  import swing._
7  import swing.event._
8  import scala.swing._
9
10 /* lib */
11 import sclack.domain.{Character, Rogue, Fighter, Wizard, GameSession}
12 import sclack.tech.TileManager
13
14 /**
15  * Dialog where we can create a particular character for playing the game.
16  * @author Simon Symeonidis
17  */
18 class CreateCharacter extends Dialog {
19   val okText       : String = "Ok"
20   val cancelText   : String = "Cancel"
21   val wizardText   : String = "Wizard"
22   val fighterText  : String = "Fighter"
23   val rogueText    : String = "Rogue"
24
25   val radioClasses : Array[String]      =
26     Array(wizardText, fighterText, rogueText)
27
28   var thelp = TileManager
```

```
29
30  var radioGroup    : ButtonGroup        = null
31  var radios        : Array[RadioButton] = new Array[RadioButton](0)
32  var radioPanel    : BoxPanel           = new BoxPanel(Orientation.Horizontal)
33  var ok            : Button             = new Button{ text = okText; enabled =
        false}
34  var cancel        : Button             = new Button{ text = cancelText }
35  var characterIco : Label               = new Label{ icon = new ImageIcon(
        thelp.tile("dun",1)) }
36
37  /* Stats */
38  var player : Character = new Wizard
39
40  var constitutionLabel : Label =
41    new Label{ text = "Constitution : " + player.combinedConstitution }
42
43  var intelligenceLabel : Label =
44    new Label{ text = "Intelligence : " + player.combinedIntelligence }
45
46  var strengthLabel     : Label =
47    new Label{ text = "Strength : " + player.combinedStrength }
48
49  var nameLabel         : Label =
50    new Label{ text = "Name : " }
51
52  var nameEdit          : TextField = new TextField()
53
54  val namePanel = new BoxPanel(Orientation.Vertical) {
55    nameEdit.maximumSize = new Dimension(250, 20)
56    contents ++= List(nameLabel, nameEdit)
57  }
58
59  var statsList : Array[Component] =
60    Array[Component](namePanel, constitutionLabel,
61                     intelligenceLabel, strengthLabel)
62
63  /* UI Setup start */
64  title = "Create your character"
65  modal = true
66
67  preferredSize = new Dimension(300,200)
68  maximumSize   = new Dimension(300,200)
69  minimumSize   = new Dimension(300,200)
70
71  /* Make the radios */
72
73  val bottomButtons = new BoxPanel(Orientation.Horizontal) {
74    contents += ok
75    contents += cancel
76  }
77
78  val classRadios = new BoxPanel(Orientation.Horizontal) {
79    for (rc <- radioClasses) radios :+= new RadioButton(rc)
80    radioGroup = new ButtonGroup(radios:_*)
81    radioGroup.select(radios.head)
82    contents ++= radios
83  }
84
85  val buttonsAndRadios = new BoxPanel(Orientation.Vertical) {
86    contents += classRadios
87    contents += bottomButtons
88  }
89
```

```
 90   val statsLabels = new BoxPanel(Orientation.Vertical){
 91     maximumSize = new Dimension(200,400)
 92     minimumSize = new Dimension(200,200)
 93     preferredSize = new Dimension(200,200)
 94     contents ++= statsList
 95   }
 96
 97   contents = new BorderPanel {
 98     import BorderPanel.Position._
 99     layout(statsLabels)      = East
100     layout(characterIco)     = Center
101     layout(buttonsAndRadios) = South
102     border = new EmptyBorder(10,10,10,10)
103   }
104
105   /* UI Setup end */
106
107   /* Reactions */
108   listenTo(radios:_*)
109   listenTo(ok, cancel, nameEdit)
110   reactions += {
111     case ButtonClicked(b) =>
112       b.text match {
113         case `okText`      => reactOnOk
114         case `cancelText`  => reactOnCancel
115         case `fighterText` => chooseFighter
116         case `wizardText`  => chooseWizard
117         case `rogueText`   => chooseRogue
118       }
119     case _ => satisfiedFormConditions
120   }
121
122   /* End Reactions */
123
124   /** Switch character, and change graphics */
125   private def chooseFighter {
126     player = new Fighter()
127     characterIco.icon = new ImageIcon(thelp.tile("dun",1))
128     update
129   }
130
131   /** Switch character, and change graphics */
132   private def chooseWizard  {
133     player = new Wizard()
134     characterIco.icon = new ImageIcon(thelp.tile("dun",2))
135     update
136   }
137
138   /** Switch character, and change graphics */
139   private def chooseRogue   {
140     player = new Rogue()
141     characterIco.icon = new ImageIcon(thelp.tile("dun",3))
142     update
143   }
144
145   /**
146    * The reaction to do on an ok click
147    * TODO GameSession object should be created here
148    */
149   private def reactOnOk {
150     var session = new GameSession()
151
152     session.character = player
```

```
153     session.map.mainChar = player
154     dispose
155
156     val gameUI = new GameUI(session)
157   }
158
159   /**
160    * The reaction to do on a cancel click
161    */
162   private def reactOnCancel {
163     dispose
164   }
165
166   /**
167    * Update the information on here
168    */
169   private def update {
170     constitutionLabel.text = "Constitution : " + player.combinedConstitution
171     strengthLabel.text     = "Strength : "     + player.combinedStrength
172     intelligenceLabel.text = "Intelligence : " + player.combinedIntelligence
173   }
174
175   /**
176    * Method to check if all the form conditions have been satisfied. In this
177    * case if the user has chosen a class, and has given the character a name
178    * longer than 3 characters
179    *
180    * @note We can use Scala's Unit here in order to feed these predicates as
181    *   parameters and actions to perform as well. Hence this behaviour can be
182    *   later on extracted as a trait.
183    */
184   private def satisfiedFormConditions {
185     ok.enabled = nameEdit.text.length >= 3
186   }
187
188   centerOnScreen()
189   open()
190 }
```

Listing 17: Create Character

```
1  package sclack.ui
2
3  import javax.swing.ImageIcon
4  import java.awt.Dimension
5  import swing._
6  import swing.event._
7  import scala.swing._
8
9  /**
10  * Menu to delete a required game.
11  *
12  * @see     DeleteGame
13  * @see     LoadGame
14  * @author Simon Symeonidis
15  */
16 object DeleteGame {
17
18 }
```

Listing 18: Delete Game

```scala
package sclack.ui

import javax.swing.ImageIcon
import java.awt.Dimension
import swing._
import swing.event._
import scala.swing._

import sclack.ui.widgets.{InfoWidget, StatsWidget, WorldWidget,
                          ActionButtons}
import sclack.domain.GameSession

/**
 * Main game user interface.
 *
 * @see     DeleteGame
 * @see     LoadGame
 * @author Simon Symeonidis
 */
class GameUI(sess: GameSession) extends Dialog {
  title = "SCLACK! SCLACK! SCLACK!"
  modal = true

  var session = sess

  /** Thanks to http://vimeo.com/13900342 (Ken Scambler) */
  contents = new BorderPanel{
    import BorderPanel.Position._

    /* Some standard initializement */
    WorldWidget.currMap   = session.map
    ActionButtons.session = session

    /* Add the components to the main ui layout */
    layout(StatsWidget) = East
    layout(WorldWidget) = Center
    layout(InfoWidget)  = South
  }

  centerOnScreen()
  open()
}
```

Listing 19: Game UI

```scala
package sclack.ui

import swing._
import scala.swing._
import swing.event._
import javax.swing.border.EmptyBorder

import sclack.ui.factories.{TextFieldFactory, TextAreaFactory}

/**
 * The Credits dialog to respect the artists (and ego whoring :D)
 * @author Simon Symeonidis
 */
class Credits extends Dialog {

  val tff = TextFieldFactory
```

```
17    val taf = TextAreaFactory
18
19    val programmingLabel = new Label{ text = "Programming: " }
20    val authorField      = tff.disabledTextField("Simon (psyomn) Symeonidis")
21    val thanksLabel      = new Label{ text = "Thanks to: " }
22    val thanksArea       = taf.disabledTextArea(
23      "Jerom for the fantasy tileset:\n"
24      + "  http://opengameart.org/sites/default/files/tileset_16x16_Jerom_CC-BY-
            SA-3.0_1.png&nid=17136\n\n"
25      + "Gwes for the NES style tileset:\n"
26      + "  http://opengameart.org/content/16x16-dungeon-tiles-nes-remake")
27    val back             = new Button{ text = "Back" }
28
29    val components : Array[Component] =
30      Array[Component](programmingLabel, authorField, thanksLabel,
31                       thanksArea, back)
32
33    title    = "Credits"
34    modal    = true
35
36    preferredSize = new Dimension(700,400)
37    maximumSize   = new Dimension(700,400)
38    minimumSize   = new Dimension(700,400)
39
40    contents = new BoxPanel(Orientation.Vertical) {
41      contents ++= components
42
43      border = new EmptyBorder(10,10,10,10)
44    }
45
46    listenTo(back)
47    reactions += {
48      case ButtonClicked(b) =>
49        dispose
50    }
51
52    centerOnScreen()
53    open()
54  }
```

Listing 20: Credits

```
1  package sclack.ui
2
3  /**
4   * Some static information that can be accessed from here from the rest of the
5   * application.
6   */
7  object Sclack {
8    val version = "0.3"
9    val authors = Array("Simon Symeonidis (psyomn)")
10   val license = "GPL v3.0"
11 }
```

Listing 21: Sclack

```
1  package sclack.ui.widgets
2
3  import scala.swing.GridBagPanel
```

```
 4
 5 import sclack.domain.Character
 6 import sclack.ui.factories.TextFieldFactory
 7
 8 /**
 9  * Per character widget that demonstrates the character's information on a UI
10  * @author Simon Symeonidis
11  */
12 class StatsView(character: Character) extends GridBagPanel {
13
14   val constraint = new Constraints
15   constraint.gridx = 0
16   constraint.gridy = 0
17   constraint.fill  = GridBagPanel.Fill.Both
18
19   val tf = TextFieldFactory
20
21   /* Index for the grid bag rows */
22   var ix = 0
23   var iy = 0
24
25   add(tf.disabledTextField("Hello there"), constraint); next
26   add(tf.disabledTextField("Stalker"), constraint); next
27   add(tf.disabledTextField("Hello there"), constraint); next
28   add(tf.disabledTextField("Stalker"), constraint); next
29   add(tf.disabledTextField("Hello there"), constraint); next
30   add(tf.disabledTextField("Stalker"), constraint); next
31
32   private def next = {
33     if (ix == 1) {
34       ix  = 0
35       iy += 1
36     }
37     else {
38       ix += 1
39     }
40     constraint.gridx = ix
41     constraint.gridy = iy
42   }
43 }
```

Listing 22: Stats View

```
 1 package sclack.ui.widgets
 2
 3 import javax.swing.border.EmptyBorder
 4 import javax.swing.ImageIcon
 5 import java.awt.Dimension
 6 import java.awt.Graphics2D
 7 import java.awt.geom.Line2D
 8 import swing._
 9 import swing.event._
10 import scala.swing._
11
12 import sclack.tech.TileManager
13
14 /**
15  * This is the widget where the world is demonstrated upon. Whatever happens
       to
16  * the domain version of the world should be represented here.
17  *
```

```
18  * @author Simon Symeonidis
19  */
20 object WorldWidget extends Panel {
21   /** The current map that we wish to render. */
22   var currMap : sclack.domain.Map = _
23
24   var tileMan = TileManager
25
26   preferredSize = new Dimension(400,400)
27   maximumSize   = new Dimension(400,400)
28   minimumSize   = new Dimension(400,400)
29   border        = new EmptyBorder(10,10,10,10)
30
31   /**
32    * Draw the required stuff on screen by overidding
33    * whatever else might have been needed...
34    */
35   override def paint(g: Graphics2D) {
36     import scala.util.Random
37
38     var rand = new Random()
39
40     drawMap(g)
41     drawNPCs(g)
42     drawPlayer(g)
43
44     g.finalize()
45   }
46
47   /* Isolate drawing logic for maps */
48   private def drawMap(g: Graphics2D) {
49     /* Should be done in a better way... */
50     val width  = currMap.width  - 1
51     val height = currMap.height - 1
52     var currentTile : Int = 0
53
54     for (i <- 0 to height){
55       for (j <- 0 to width){
56         currentTile = (height + 1) * i + j
57         g.drawImage(
58           tileMan.tile("fan",
59           currMap.at(i,j)), /* What to draw */
60           j * 16, i * 16, null)
61       }
62     }
63   }
64
65   /* Isolate drawing logic for maps */
66   private def drawNPCs(g: Graphics2D) {
67     for (ent <- currMap.entities){
68       g.drawImage(ent._3.demonstrate, ent._1, ent._2, null)
69     }
70   }
71
72   /* For drawing the main player on the map */
73   private def drawPlayer(g: Graphics2D) {
74      g.drawImage(
75        currMap.mainChar.demonstrate,
76        currMap.mainCharPos._1 * 16,
77        currMap.mainCharPos._2 * 16,
78        null)
79   }
80 }
```

Listing 23: World Widget

```
1  package sclack.ui.widgets
2
3  import javax.swing.border.EmptyBorder
4  import javax.swing.ImageIcon
5  import java.awt.Dimension
6  import swing._
7  import swing.event._
8  import scala.swing._
9
10 /**
11  * The info widget of the main window. The user may read information about
        what
12  * the character is doing more precisely in the text displayed in this widget.
13  * There are other facilities available in order to make this more usable
14  * (clear text etc)
15  *
16  * @author Simon Symeonidis
17  */
18 object InfoWidget extends BorderPanel {
19   val tmpButts : Array[String] = Array("Clear", "Derp", "Herp", "Derpa")
20   val buttons = new BoxPanel(Orientation.Horizontal) {
21     for (s <- tmpButts)
22       contents += new Button{ text = s }
23   }
24
25   preferredSize = new Dimension(600,200)
26   maximumSize   = new Dimension(600,200)
27   minimumSize   = new Dimension(600,200)
28   border        = new EmptyBorder(10,10,10,10)
29
30   val infoList : ListView[String] = new ListView[String]
31   infoList.fixedCellHeight = 100
32   infoList.fixedCellWidth  = 100
33
34   import BorderPanel.Position._
35   layout(infoList) = Center
36   layout(buttons)  = South
37 }
```

Listing 24: InfoWidget

```
1  package sclack.ui.widgets
2
3  import javax.swing.border.EmptyBorder
4  import javax.swing.ImageIcon
5  import java.awt.Dimension
6  import swing._
7  import swing.event._
8  import scala.swing._
9  import GridBagPanel._
10 import Array._
11
12 import sclack.domain.GameSession
13 import sclack.tech.CoordinateHelper
14
```

```
15 /**
16  * The buttons that will help to do different actions
17  *
18  * @author
19  */
20 object ActionButtons extends GridBagPanel {
21   val buttonsCaptions : Array[String] = Array[String](
22     "?",       "/\\",     "?",
23     "<",       "center",  ">",
24     "Attack", "V",       "Observe")
25
26   var session : GameSession = _
27
28   val buttCapNorth    = buttonsCaptions(1)
29   val buttCapSouth    = buttonsCaptions(7)
30   val buttCapEast     = buttonsCaptions(5)
31   val buttCapWest     = buttonsCaptions(3)
32   val buttCapAttack   = buttonsCaptions(6)
33   val buttCapObserve  = buttonsCaptions(8)
34   val buttCapCenter   = buttonsCaptions(4)
35   val buttCapSpecial1 = buttonsCaptions(0)
36   val buttCapSpecial2 = buttonsCaptions(2)
37
38   val constraints = new Constraints
39   var ix = 0
40   var iy = 0
41
42   val buttons : Array[Button] = buttonsCaptions.map(new Button(_))
43
44   /** Add the action buttons to the layout */
45   constraints.fill = Fill.Horizontal
46   for(x <- 0 to 8) {
47     constraints.gridx = x % 3
48     constraints.gridy = (x / 3.0).toInt
49     add(buttons(x), constraints)
50   }
51
52   listenTo(buttons:_*)
53   reactions += {
54     case ButtonClicked(b) =>
55       b.text match {
56         case `buttCapSpecial1` => reactSpecial1
57         case `buttCapSpecial2` => reactSpecial2
58         case `buttCapNorth`    => reactNorth
59         case `buttCapWest`     => reactWest
60         case `buttCapCenter`   => reactCenter
61         case `buttCapEast`     => reactEast
62         case `buttCapAttack`   => reactAttack
63         case `buttCapSouth`    => reactSouth
64         case `buttCapObserve`  => reactObserve
65       }
66
67     WorldWidget.repaint
68   }
69
70   private def reactNorth =
71     session.map.mainCharPos =
72       CoordinateHelper.moveNorth(session.map.mainCharPos)
73
74   private def reactSouth =
75     session.map.mainCharPos =
76       CoordinateHelper.moveSouth(session.map.mainCharPos)
77
```

```
78    private def reactEast =
79      session.map.mainCharPos =
80        CoordinateHelper.moveEast(session.map.mainCharPos)
81
82    private def reactWest =
83      session.map.mainCharPos =
84        CoordinateHelper.moveWest(session.map.mainCharPos)
85
86    private def reactCenter =println("center")
87    private def reactAttack =println("attack")
88    private def reactObserve =println("observe")
89    private def reactSpecial1 =println("spec1")
90    private def reactSpecial2 =println("spec2")
91  }
```

Listing 25: Action Buttons

```
1  package sclack.ui.widgets
2
3  import javax.swing.border.EmptyBorder
4  import javax.swing.ImageIcon
5  import java.awt.Dimension
6  import swing._
7  import swing.event._
8  import scala.swing._
9  import Array._
10
11 import sclack.domain.Rogue
12
13 /**
14  * This is where all the stats information of the character are put, along
        with
15  * dialogs in order to add equipment, use items, etc.
16  *
17  * @author Simon Symeonidis
18  */
19 object StatsWidget extends BorderPanel {
20
21   val characterStats = new BoxPanel(Orientation.Vertical){
22   }
23
24   /**
25    * @note Thank to
26    *   http://www.scala-lang.org/old/node/2896 for explaining the cryptic use
27    *   of tabbed panes in scala-swing ...
28    */
29   val tabs : TabbedPane = new TabbedPane {
30     pages += new TabbedPane.Page("Stats", new StatsView(new Rogue()))
31     pages += new TabbedPane.Page("Inventory", new Label{text= "Inventory"})
32   }
33
34   /* Widget Dimensions */
35   preferredSize = new Dimension(400,100)
36   maximumSize   = new Dimension(400,100)
37   minimumSize   = new Dimension(400,100)
38   border        = new EmptyBorder(10,10,10,10)
39
40   /* Add the components here */
41   import BorderPanel.Position._
42   layout(tabs)          = Center
43   layout(ActionButtons) = South
```

```
44
45 }
```

Listing 26: Statistics Widget

```
1  package sclack.ui.factories
2
3  import scala.swing.TextArea
4  import java.awt.Color
5
6  object TextAreaFactory {
7    def disabledTextArea(t: String) : TextArea = {
8      new TextArea{
9          text       = t
10         editable   = false
11         background = new Color(200,200,200)
12     }
13   }
14 }
```

Listing 27: Text Area Factory (UI widget helper)

```
1  package sclack.ui.factories
2
3  import scala.swing.TextField
4  import java.awt.Color
5
6  /**
7   * For creating text fields with special settings quickly
8   * @author Simon Symeonidis
9   */
10 object TextFieldFactory {
11
12   /**
13    * Create a disabled (grayed out) text field with the required
14    * information inside
15    *
16    * @param t
17    *   is the string to set to the text field
18    * @return
19    *   a grayed out text field with the specified contents
20    */
21   def disabledTextField(t: String) : TextField = {
22     new TextField{
23         text       = t
24         editable   = false
25         background = new Color(200,200,200)
26     }
27   }
28 }
```

Listing 28: lst:ui:tff

```
1  package sclack.ui
2
3  import javax.swing.border.EmptyBorder
4  import javax.swing.ImageIcon
```

```
 5 import java.awt.Dimension
 6 import swing._
 7 import swing.event._
 8 import scala.swing._
 9
10 import sclack.domain.GameSession
11 import sclack.domain.commands.{CreateNewGame, DeleteGame, LoadGame}
12
13 /**
14  * The main menu of the application. This is where the user specifies whether
15  * we want a new game, to load a game, or to delete a game.
16  *
17  * @author Simon Symeonidis
18  */
19 object MainMenu extends SimpleSwingApplication {
20
21   val newGameText      = "New Game"
22   val loadGameText     = "Load Game"
23   val deleteGameText   = "Delete Game"
24   val quitText         = "Quit"
25   val creditsText      = "Credits"
26   val imageLocation    = getClass.getResource("/title.png")
27
28   var newGameButton    = new Button {text = newGameText}
29   var loadGameButton   = new Button {text = loadGameText}
30   var deleteGameButton = new Button {text = deleteGameText}
31   var quitGameButton   = new Button {text = quitText}
32   var creditsButton    = new Button {text = creditsText}
33   var logoIcon         = new ImageIcon(imageLocation)
34   var labelIcon        = new Label{icon = logoIcon}
35
36   def top = new MainFrame {
37     title = "Sclack v" + Sclack.version
38
39     contents = new BoxPanel(Orientation.Vertical){
40       contents += labelIcon
41       contents += newGameButton
42       contents += loadGameButton
43       contents += deleteGameButton
44       contents += creditsButton
45       contents += quitGameButton
46       border    = new EmptyBorder(10,10,10,10)
47     }
48   }
49
50   listenTo(newGameButton, loadGameButton, deleteGameButton, quitGameButton,
51            creditsButton)
52   reactions += {
53     case ButtonClicked(b) =>
54       b.text match {
55          case `newGameText`    => createNewGame
56          case `loadGameText`   => loadGame
57          case `deleteGameText` => deleteGame
58          case `quitText`       => quitGame
59          case `creditsText`    => creditsDialog
60       }
61   }
62
63   /**
64    * React to the create game click, by creating a new game session
65    */
66   private def createNewGame {
67     val characterUI = new CreateCharacter()
```

```
68    }
69
70    /**
71     * React to the load game click, by load a new game session
72     */
73    private def loadGame {
74      new GenericGameSelector("Load", new LoadGame())
75    }
76
77    /**
78     * React to the delete game click, by delete a new game session
79     */
80    private def deleteGame {
81      new GenericGameSelector("Delete", new DeleteGame())
82    }
83
84    /**
85     * Open the credits dialog
86     */
87    private def creditsDialog {
88      new Credits()
89    }
90
91    /**
92     * Quit the application
93     * TODO need to fix this properly
94     */
95    private def quitGame {
96      println("Bye.")
97      quit()
98    }
99  }
```

Listing 29: Main Menu of the game

5.2 Domain

```
1  package sclack.domain
2
3  /**
4   * A fighter class that is proficient in melee combat.
5   * @author Simon Symeonidis
6   */
7  class Fighter extends Character {
8    override def observe = "You see a muscular, intimidating person."
9    def combinedStrength = strength + 5
10   def combinedIntelligence = intelligence - 3
11   def combinedConstitution = constitution + 2
12   def combinedDexterity = dexterity + 1
13   def discipline = "Fighter"
14 }
```

Listing 30: Figher Class

```
1  package sclack.domain
2
3  class Monster extends Character{
```

```
4   def combinedStrength = strength + 1
5   def combinedIntelligence = intelligence + 3
6   def combinedConstitution = constitution + 1
7   def combinedDexterity = dexterity + 10
8   def discipline = "WARRRG ROOAR GWARRG!!"
9 }
```

Listing 31: Monster Class

```
1 package sclack.domain
2
3 import sclack.tech.TileManager
4
5 /**
6  * Characters are the in game characters that you can use (for example
       fighters
7  * wizards, rogues etc).
8  *
9  * @author Simon Symeonidis
10 */
11 abstract class Character extends Entity with Observable with Demonstratable{
12
13   def observe = "A fine fellow"
14
15   def demonstrate = TileManager.tile("dun", 3)
16
17   /** Hitpoints are the current life of the character */
18   var hitpoints    : Int = 10
19
20   /** Constitution is the max life of the character */
21   var constitution : Int = 10
22
23   /** Intelligence of the character */
24   var intelligence : Int = 5
25
26   /** Current magic points of the character */
27   var magicPoints  : Int = 0
28
29   /** Strength of the character */
30   var strength     : Int = 1
31
32   /** Dexterity */
33   var dexterity    : Int = 1
34
35   /** Skillpoints that may be used in order to improve stats */
36   var skillpoints  : Int = 0
37
38   /** Somewhat cosmetic thing that shows us the current level */
39   var level        : Int = 1
40
41   /** The experience points of the character */
42   var experience   : Int = 0
43
44   /** TODO maybe some other formula for this one */
45   def levelUp {
46     skillpoints += 2 * level
47   }
48
49   /** Interfacing method to the actual back-end method for safe increase */
50   def improveConstitution = increaseConstitution
51
```

```
52    /** Interfacing method to the actual back-end method for safe increase */
53    def improveIntelligence = increaseIntelligence
54
55    /** Interfacing method to the actual back-end method for safe increase */
56    def improveStrength     = increaseStrength
57
58    /** Class + Armor combination for stat */
59    def combinedStrength : Int
60
61    /** Class + Armor combination for stat */
62    def combinedIntelligence : Int
63
64    /** Class + Armor combination for stat */
65    def combinedConstitution : Int
66
67    /** Class + Armor combination for stat */
68    def combinedDexterity : Int
69
70    /**
71     * Easy way to discern class (should not be used programmatically for
72     * checks)
73     */
74    def discipline : String
75
76    /**
77     * Safely increase ability
78     */
79    private def increaseStrength {
80      if (enoughSkillpoints(strength)){
81        skillpoints -= strength
82        strength    += 1
83      }
84    }
85
86    /**
87     * Safely increase ability
88     */
89    private def increaseIntelligence {
90      if (enoughSkillpoints(intelligence)){
91        skillpoints  -= intelligence
92        intelligence += 1
93      }
94    }
95
96    /**
97     * Safely increase ability
98     */
99    private def increaseConstitution {
100     if (enoughSkillpoints(constitution)){
101       skillpoints  -= constitution
102       constitution += 1
103     }
104   }
105
106   /**
107    * Quickhand to check if the player has enough skillpoints to increase a
108    * given skill
109    */
110   private def enoughSkillpoints(abilityAmount : Int) : Boolean =
111     skillpoints >= abilityAmount;
112
113 }
```

Listing 32: Character Class

```
1 package sclack.domain
2
3 /**
4  * Superclass for items.
5  * @author Simon Symeonidis
6  */
7 class Item {
8
9 }
```

Listing 33: Item Class

5.2.1 Commands

```
1 package sclack.domain.commands
2
3 /**
4  * Transaction that holds everything when using an item in game.
5  * @author Simon Symeonidis
6  */
7 class UseItem extends Commandable {
8   /** Use the item, waste a turn */
9   def execute() {
10  }
11 }
```

Listing 34: Use Item

```
1 package sclack.domain.commands
2
3 import sclack.domain.GameSession
4 import sclack.domain.Character
5
6 /**
7  * Create new game command. This should create a new game, and return the
8  * GameSession object
9  * @author Simon Symeonidis
10 */
11 class CreateNewGame(character : Character) extends Commandable {
12   def execute {
13     gameSession = new GameSession()
14   }
15
16   var gameSession : GameSession = _
17 }
```

Listing 35: Create New Game

```
1 package sclack.domain.commands
```

```
2
3 /**
4  * Trait that describes the Command Pattern
5  * @author Simon Symeonidis
6  */
7 trait Commandable {
8   def execute
9 }
```

Listing 36: Commandable Trait

```
1 package sclack.domain.commands
2
3 /**
4  * For loading a new game
5  * @author Simon Symeonidis
6  */
7 class LoadGame extends Commandable {
8   def execute {
9     println("Load Game Command Executed")
10  }
11 }
```

Listing 37: Load Game

```
1 package sclack.domain.commands
2
3 /**
4  * For deleting a previous game
5  * @author Simon Symeonidis
6  */
7 class DeleteGame extends Commandable {
8   def execute {
9     println("Delete Game Command Executed")
10  }
11 }
```

Listing 38: Delete Game

```
1 package sclack.domain
2
3 /**
4  * Enumeration for the different elements that exist.
5  * @author Simon Symeonidis
6  */
7 object Element extends Enumeration {
8   type Element = Value
9   val  fire, water, wind, earth, physical,
10       void, shadow, lightning, sonic = Value
11 }
```

Listing 39: Element

```
1 package sclack.domain
```

```
 2
 3 /**
 4  * This is a user class that is to help encapsulate information about the
 5  * current person using the system - not to be confused with the actual in
 6  * game characters.
 7  *
 8  * @author Simon Symeonidis
 9  */
10 class User(n: String){
11
12   /** The identity of the user */
13   val id : Long = -1
14
15   def name = n
16 }
```

Listing 40: User

```
 1 package sclack.domain
 2
 3 /**
 4  * Trait for things that are fightable
 5  * @author Simon Symeonidis
 6  */
 7 trait Fightable {
 8   def attack
 9   def defend
10   def cast
11 }
```

Listing 41: Fightable

```
 1 package sclack.domain
 2
 3 import sclack.domain.factories.MapFactory
 4
 5 /**
 6  * Encapsulate the information for a single game session
 7  * @author Simon Symeonidis
 8  */
 9 class GameSession {
10   /** The profile of the user. We're not really using this at the moment */
11   var user      : User = _
12
13   /** The characters that the user has */
14   var character : Character = _
15
16   /** The seed of the current world */
17   var seed      : Long = -1
18
19   /** The world that the user interacts with. If we were to implement the
20       application fully, then we would be using this. But for now we just
21       use a single map */
22   var world     : World = _
23
24   /** The map that the game takes place on */
25   var map       : Map   = MapFactory.createSingleMap
26
27   map.mainChar = character
```

```
28 }
```

Listing 42: Game Session

```
1  package sclack.domain
2
3  import scala.collection.mutable.Queue
4
5  /**
6   * The domain object that represents the game history / actions that have
7   * happened in the last session.
8   * @author Simon Symeonidis
9   */
10 class GameHistory {
11
12   def log(l: String){
13     if (logs.size + 1 > max)
14       logs.dequeue
15     logs.enqueue(l)
16   }
17
18   def count : Int = logs.size
19
20   /** the maximum amount of logs to keep */
21   private var max  = 100
22
23   /** the log data structure we're using to perform this */
24   private var logs : Queue[String] = new Queue[String]
25 }
```

Listing 43: Game History

```
1  package sclack.domain
2
3  import javax.swing.ImageIcon
4  import java.awt.image.BufferedImage
5
6  /**
7   * Create npcs that say random stuff when you talk to them this way.
8   *
9   * @author Simon Symeonidis
10  */
11 class NonPlayableCharacter(speech: String, ico: BufferedImage)
12       extends Entity with Demonstratable {
13   def observe     = speech
14   def demonstrate = ico
15 }
```

Listing 44: Non Playable Character

```
1  package sclack.domain
2
3  /**
4   * Class that encapsulates the behaviour of a badass wizard. The wizard is
5   * wimpy, but wait until he casts his spells!
6   * @author Simon Symeonidis
7   */
```

```
 8 class Wizard extends Character {
 9   override def observe = "You see a scholar looking fellow"
10   def combinedStrength = strength - 1
11   def combinedIntelligence = intelligence + 5
12   def combinedConstitution = constitution - 1
13   def combinedDexterity = dexterity - 1
14   def discipline = "Wizard"
15 }
```

Listing 45: Wizard

```
1 package sclack.domain
2
3 class World {
4
5 }
```

Listing 46: World

```
 1 package sclack.domain
 2
 3 import scala.util.Random
 4
 5 /**
 6  * A die class to be used in conjuction with attacks, spells, etc.
 7  * specifications
 8  * @author Simon Symeonidis
 9  */
10 class Die(sides: Int){
11
12   /**
13    * Roll the dice
14    * @return A random number based on the number of sides specified
15    */
16   def roll = random.nextInt.abs % sides + 1
17
18   /** Delegate randomness to random random object.*/
19   val random = new Random()
20 }
```

Listing 47: Die

```
 1 package sclack.domain
 2
 3 import java.awt.image.BufferedImage
 4
 5 /**
 6  * This trait is purely for the gui aspect of things. If something is
 7  * demonstratable then there exists (or at least should exist) a valid name
 8  * of the tileset, and an id which can be used to map to that particular
 9  * graphical region.
10  *
11  * @author Simon Symeonidis
12  */
13 trait Demonstratable {
14   def demonstrate : BufferedImage
15 }
```

Listing 48: Demonstratable Trait

```
1  package sclack.domain
2
3  /**
4   * Rogues are the Ninjas of the fantasy world! Better not make one angry, or
5   * you won't be sleeping alone tonight...
6   * @author Simon Symeonidis
7   */
8  class Rogue extends Character {
9    override def observe = "You see a ninja like person"
10
11   def combinedStrength     = strength + 1
12   def combinedIntelligence = intelligence + 3
13   def combinedConstitution = constitution + 1
14   def combinedDexterity    = dexterity + 10
15   def discipline = "Rogue"
16 }
```

Listing 49: Rogue

```
1  package sclack.domain
2
3  /**
4   * A map is a smaller part of the World object.
5   *
6   * @author Simon Symeonidis
7   * @see World
8   */
9  class Map {
10
11   def at(x: Int, y: Int) = data(x)(y)
12
13   def width  : Int = data.head.length
14
15   def height : Int = data.length
16
17   var data : Array[Array[Int]] = Array.fill[Int](25,25){4}
18
19   var obstructions = Array.fill[Int](25,25){0}
20
21   var entities : Array[(Int, Int, NonPlayableCharacter)] = Array()
22
23   var mainChar : Character = _
24
25   /* Default position whenever the player starts playing the game */
26   var mainCharPos : (Int, Int) = (1, 1)
27 }
```

Listing 50: Map

```
1  package sclack.domain
2
3  /**
```

```
4  * This trait is for things that are observable. This will return a string
       that
5  * should describe what the resource that we are observing looks like.
6  *
7  * @author Simon Symeonidis
8  */
9 trait Observable {
10   def observe : String
11 }
```

Listing 51: Observable

```
1 package sclack.domain
2
3 /**
4  * A portal is a gateway from one map to another. This helps keep the code
5  * cleaner by delegating the actions of warping a character from one part to
6  * another to a single class.
7  *
8  * @author Simon Symeonidis
9  */
10 class Portal extends Observable {
11   def observe = "You see a gateway here that leads to ... "
12 }
```

Listing 52: Portal

```
1 package sclack.domain
2
3 /**
4  * The specific exception class for this application
5  * @author Simon Symeonidis
6  */
7 class SclackException(message: String) extends Exception {
8 }
```

Listing 53: Sclack Exception

```
1 package sclack.domain
2
3 /**
4  * Store basic application information about Sclack here (authors, versions
5  * etc).
6  *
7  * @author Simon Symeonidis
8  */
9 object Sclack {
10 }
```

Listing 54: Sclack

```
1 package sclack.domain
2
3 /**
```

```
 4  * An entity may be anything on the map.
 5  * @author Simon Symeonidis
 6  */
 7 trait Entity {
 8   /** Position on the map */
 9   var position_x : Int = -1
10
11   /** Position on the map */
12   var position_y : Int = -1
13
14   /** @return a String that describes what the entity looks like */
15   def observe : String
16 }
```

Listing 55: Entity

```
 1 package sclack.domain
 2
 3 import sclack.domain.Element._
 4
 5 /**
 6  * Class that represents damage to be dealt between enemies.
 7  * @author Simon Symeonidis
 8  */
 9 class Damage(amount: Int, element: Element){
10   def amt = amount
11   def elmnt = element
12 }
```

Listing 56: Damage

```
 1 package sclack.domain.factories
 2
 3 import sclack.domain.{Character, Fighter, Wizard, Rogue}
 4
 5 /**
 6  * Factory class that helps create Characters, because the creation is a
 7  * complex process.
 8  *
 9  * @author Simon Symeonidis
10  */
11
12 object CharacterFactory  {
13   /** Create a wizard with random stats */
14   def createWizard : Wizard = {
15     var wizard = new Wizard()
16     return wizard
17   }
18
19   /** Create a fighter with random stats */
20   def createFighter : Fighter = {
21     var fighter = new Fighter()
22     return fighter
23   }
24
25   /** Create a rogue with random stats */
26   def createRogue : Rogue = {
27     var rogue = new Rogue
28     return rogue
```

```
29   }
30 }
```

Listing 57: Character Factory

```
1  package sclack.domain.factories
2
3  import javax.swing.ImageIcon
4
5  import sclack.domain.Map
6  import sclack.domain.NonPlayableCharacter
7  import sclack.domain.Entity
8
9  import sclack.tech.TileManager
10
11 /**
12  * Factory for creating various maps that we may or may not use. I know that
13  * this class looks horrible due to all the hard coded arrays, but this is
14  * ok for now. If this were to be a 'good' game and have things programmed
15  * properly, assets would be loaded from a separate location. I don't have
16  * the luxury of time to do this like that however, so this will have to do.
17  *
18  * @author Simon Symeonidis
19  */
20 object MapFactory {
21
22   /** Create a map with no portals */
23   def createSingleMap : sclack.domain.Map = {
24     var map : sclack.domain.Map = new sclack.domain.Map()
25
26     map.data = Array[Array[Int]](
27       Array(140, 131, 131, 131, 131, 131, 131, 131, 131, 131, 131, 131,
28             131, 131, 131, 131, 131, 131, 131, 131, 131, 131, 140),
29       Array(140, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120,
30             120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 140),
31       Array(140, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120,
32             120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 140),
33       Array(140, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120,
34             120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 140),
35       Array(140, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120,
36             120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 140),
37       Array(140, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120,
38             120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 140),
39       Array(140, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120,
40             120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 140),
41       Array(140, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120,
42             120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 140),
43       Array(140, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120,
44             120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 140),
45       Array(140, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120,
46             120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 140),
47       Array(140, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120,
48             120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 140),
49       Array(140, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120,
50             120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 140),
51       Array(140, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120,
52             120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 140),
53       Array(140, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120,
54             120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 140),
55       Array(140, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120,
56             120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 140),
```

```
57        Array(140, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120,
58              120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 140),
59        Array(140, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120,
60              120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 140),
61        Array(140, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120,
62              120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 140),
63        Array(140, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120,
64              120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 140),
65        Array(140, 120, 120, 120, 120, 120, 117, 117, 117, 120, 120, 120,
66              120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 140),
67        Array(140, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120,
68              120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 140),
69        Array(140, 120, 120, 120, 120, 120, 116, 120, 120, 120, 120, 120,
70              120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 140),
71        Array(140, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120,
72              120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 140),
73        Array(140, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120,
74              120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 140),
75        Array(131, 131, 131, 131, 131, 131, 131, 131, 131, 131, 131, 131,
76              131, 131, 131, 131, 131, 131, 131, 131, 131, 131, 131, 131))
77
78      var npcs = createGenericNPCs
79
80      map.obstructions = createGenericObstructions
81
82      map.entities = Array[(Int, Int, NonPlayableCharacter)](
83        (160         , 160, npcs(0)),
84        (160 + 1 * 16, 160, npcs(1)),
85        (160 + 2 * 16, 160, npcs(2)),
86        (160 + 3 * 16, 160, npcs(3)),
87        (160 + 4 * 16, 160, npcs(4)))
88
89      for(ent <- map.entities)
90        map.obstructions(ent._2 / 16)(ent._1 / 16) = 1
91
92      map
93    }
94
95
96    /**
97     * Create generic NPCs. This is mainly for testing out speech capabilities,
98     * etc.
99     *
100    * @return an array of non playable characters.
101    */
102   def createGenericNPCs : Array[NonPlayableCharacter] = {
103     Array[String](
104       "Oh, why hello there!",
105       "Don't mind us, we're just some random NPCs for testing",
106       "Hopefully we'll be included in the real game one day!",
107       "Actually it would be nice to see this thing be finished at some point",
108       "Hi, my name is Harry and I'M GOING TO KILL YOU"
109     ).map(new NonPlayableCharacter(_,TileManager.tile("fan",110)))
110   }
111
112   /**
113    * Simple method to create an array representing where the player can and
114    * cannot go.
115    *
116    * @return Array representing the obstructions. 1 is for obstruction, 0 is
117    *   for free things.
118    */
119   def createGenericObstructions : Array[Array[Int]] = {
```

```
120     Array[Array[Int]](
121       Array[Int](1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
122       Array[Int](1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1),
123       Array[Int](1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1),
124       Array[Int](1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1),
125       Array[Int](1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1),
126       Array[Int](1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1),
127       Array[Int](1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1),
128       Array[Int](1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1),
129       Array[Int](1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1),
130       Array[Int](1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1),
131       Array[Int](1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1),
132       Array[Int](1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1),
133       Array[Int](1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1),
134       Array[Int](1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1),
135       Array[Int](1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1),
136       Array[Int](1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1),
137       Array[Int](1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1),
138       Array[Int](1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1),
139       Array[Int](1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1),
140       Array[Int](1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1),
141       Array[Int](1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1),
142       Array[Int](1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1),
143       Array[Int](1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1),
144       Array[Int](1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1),
145       Array[Int](1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1))
146   }
147 }
```

Listing 58: Map Factory

5.3 Tech

```
1  package sclack.tech
2
3  /**
4   * Helper class that should help detect where to put different things of the
5   * running application. For example if we want to get a path in which to store
6   * the save games, this class should be responsible
7   *
8   * @author Simon Symeonidis
9   */
10 object DynamicConfiguration {
11   val home = System.getProperty("user.home")
12   val conf = home + ".config"
13   val app  = conf + "/sclack"
14   val data = app  + "/data"
15   val logs = app  + "/logs"
16 }
```

Listing 59: Dynamic Configuration

```
1  package sclack.tech
2
3  /**
4   * The database registry that should store the objects to be persisted...
5   * @author Simon Symeonidis
```

```
 */
class DbRegistry {

}
```

Listing 60: Database Registry

```
package sclack.tech

import java.awt.image.BufferedImage
import scala.collection.mutable.HashMap
import java.awt.image.BufferedImage

/**
 * The tile manager is a Cache-like pattern that only loads the required
 * tilesets once, and then they are loaded from a single point in memory when
 * required, when painting the tiles anywhere.
 *
 * @author Simon Symeonidis
 */
object TileManager {

  /**
   * The tilemap to contain the required graphics
   */
  private var tilemap = new HashMap[String,BufferedImage]

  /**
   * Get the tileset by giving the required name
   * @param name is the name of the tileset to use (for now 'dun', and 'fan'
   *   are the only valid choices
   * @param ix is the index of the tile to fetch and return
   * @return the tile that you require, given the id
   * @note Ultimately you'd
   */
  def tile(name: String, ix: Int) : BufferedImage = {
    name match {
    /* Dungeon */
    case "dun" =>
      return dungeonTileHelper.tile(ix)

    /* Fantasy */
    case "fan" =>
      return fantasyTileHelper.tile(ix)
    }
  }

  /**
   * Get the width of a particular tileset
   *
   * @param name is the name of the tileset we want. You can either specify
   *   "dun" or "fan" for the dungeon or fantasy tilesets, respectively.
   * @return the width of that particular tileset
   */
  def widthOf(name: String) : Int = {
    name match {
    case "dun" => return dungeonTileHelper.width
    case "fan" => return fantasyTileHelper.width
    }
  }

```

```
55    /**
56     * Get the height of a particular tileset
57     *
58     * @param name is the name of the tileset we want. You can either specify
59     *   "dun" or "fan" for the dungeon or fantasy tilesets, respectively.
60     * @return the height of that particular tileset.
61     */
62    def heightOf(name: String) : Int = {
63      name match {
64      case "dun" => return dungeonTileHelper.height
65      case "fan" => return fantasyTileHelper.height
66      }
67    }
68
69    private val dungeonTilesetName = "/16x16-dungeon-tiles-nes-remake.png"
70    private val fantasyTilesetName = "/16x16-fantasy-tileset.png"
71    private val dungeonTilesetRes  = getClass.getResource(dungeonTilesetName)
72    private val fantasyTilesetRes  = getClass.getResource(fantasyTilesetName)
73
74    private val dungeonTileHelper  = new TileHelper(16,16,2,dungeonTilesetRes)
75    private val fantasyTileHelper  = new TileHelper(16,16,0,fantasyTilesetRes)
76  }
```

Listing 61: Tile Manager

```
1  package sclack.tech
2
3  import java.io.File
4  import javax.imageio.ImageIO
5  import java.awt.image.BufferedImage
6  import java.awt.Graphics2D
7
8
9  /**
10  * This class should not be used directly. You should look at the TileManager
11  * class for more information.
12  *
13  * The tile helper reads an image file and extracts tiles from them. You
        create
14  * this object by specifying the tile dimentions (in our case 16x16), and then
15  * provide an index in order to select the wanted tile.
16  *
17  * @author Simon Symeonidis
18  * @see    TileManager
19  * @param  x is the width of the tile from the given tileset
20  * @param  y is the height of the tile from the given tileset
21  * @param  tileset is the name of the wanted tileset. Please set to 'dungeon'
22  *   or 'fantasy', for what you want.
23  */
24 class TileHelper(x: Int, y: Int, bdr: Int = 0, tset: java.net.URL){
25
26   lazy val tileset = ImageIO.read(tset)
27
28   /**
29    * Get a tile by getting an id
30    *
31    * @param id is the id that is specified
32    * @note Thanks to:
33    *   http://stackoverflow.com/questions/299495/ for loading buffered images
34    * @throws SclackTechnicalException if an id of 0 is given (we assume that
35    *   ids are non-zero, positive numbers)
```

```
36     */
37   def tile(id: Int) : BufferedImage = {
38     if (id <= 0) throw new SclackTechnicalException("ids are > 0")
39
40     tileset.getSubimage(
41       relativeX(id),
42       relativeY(id),
43       width,
44       height)
45   }
46
47   /**
48    * Relative X, given the id of the tile, and the border
49    *
50    * @return an integer giving the relative X to start the canvas
51    */
52   private def relativeX(id: Int) : Int =
53     ((width + border) * id) % tileset.getWidth + border
54
55   /**
56    * Relative Y, given the id of the tile, and the border
57    *
58    * @return an integer giving the relative Y to start the canvas
59    */
60   private def relativeY(id: Int) : Int =
61     (height + border) *
62     ((id - 1) / tilesPerRow).floor.toInt + border
63
64   /**
65    * The tiles we can have per row
66    *
67    * @return the number of tiles per row
68    */
69   private def tilesPerRow = tileset.getWidth / width + border
70
71   def width   = x
72   def height  = y
73   def border  = bdr
74 }
```

Listing 62: Tile Helper

```
1 package sclack.tech
2
3 /**
4  * The specific exception class for this application in the technical domain
5  * @author Simon Symeonidis
6  */
7 class SclackTechnicalException(message: String) extends Exception {
8 }
```

Listing 63: Sclack Technical Exception

```
1 package sclack.tech
2
3 /**
4  * Just a class to help with coordinates and reduce code clutter.
5  *
6  * @author Simon Symeonidis
```

```
 7  */
 8 object CoordinateHelper {
 9
10   /**
11    * Call this when you want coordinates for doing things (who is around me?
12    * what is around me?)
13    *
14    * @param  coord is the current coordinate in tuple form (x, y)
15    * @return tuples of (x, y), each containing relative points to (x, y) in
16    *   directions left, right, top, bottom.
17    */
18   def adjacentCoordinates(coord: (Int,Int)) : Array[(Int,Int)] = {
19     Array[(Int,Int)](
20       (coord._1 - 1, coord._2), /* left  */
21       (coord._1 + 1, coord._2), /* right */
22       (coord._1, coord._2 - 1), /* up    */
23       (coord._1, coord._2 + 1)  /* down  */
24     )
25   }
26
27   def moveNorth(curr: (Int,Int)) : (Int, Int) = {
28     (curr._1, curr._2 - 1)
29   }
30
31   def moveSouth(curr: (Int,Int)) : (Int, Int) = {
32     (curr._1, curr._2 + 1)
33   }
34
35   def moveEast(curr: (Int,Int)) : (Int, Int) = {
36     (curr._1 + 1, curr._2)
37   }
38
39   def moveWest(curr: (Int,Int)) : (Int, Int) = {
40     (curr._1 - 1, curr._2)
41   }
42 }
```

Listing 64: Coordinate Helper

References

[1] Philipp Haller Michel Schinz. A Scala Tutorial for Java Programmers, May 24 2011.

[2] scala lang.org. Scala API, Wed 27 Nov 2013.

[3] scala lang.org. A Tour of Scala: Pattern Matching, Wed 27 Nov 2013.