# Next Assignment:
# Simple Text Adventure Game
# (STAG)

COMSM0103

Dr Simon Lock & Dr Sion Hannuna
[ Team Si(m)on ]

# Overview

Next assignment is worth 35% of the unit grade

(it WILL contribute to your unit mark)

It is a fairly complex piece of work...

So worth spending some time getting to know !

# Overview

The purpose of this assignment is to build...
A general-purpose socket-server game-engine
(for text adventure games)

Sound familiar ?
Well it should !

This exercise is designed to "build on top of"
the experience you gained during imperative !

# Configurable Gameplay

Gameplay should be configurable by providing two "game description" files to the game engine:

- Entities: structural layout & relationships
- Actions: dynamic behaviours of the entities

We clearly need a mechanism to store this data

Because the two types of data are different in nature
We have chosen two different documents formats...

# Document Formats

DOT

A language for expressing graphs

(which is what a text adventure game is !)

entities

JSON

A language for expressing structured data

actions

We thought it would be "fun"

for you to write your own parsers !

But then we thought, perhaps not :o)

# Parsers

You've already experienced writing parsers
We don't want to cover old ground
So you'll be using two existing parsing libraries !

There is considerable educational value in
learning to use existing libraries and frameworks

This can get lost in a desire to learn fundamentals
But not on this unit !

# Which Parsers

For parsing DOT you should use "JPGD":
http://www.alexander-merz.com/graphviz/doc.html

For parsing JSON you should use "json-simple":
http://alex-public-doc.s3.amazonaws.com/json_simple-1.1/

JAR files containing these libraries are provided
(As part of a template to help you get started)

Think of this assignment as "the other half"
of the parser exercise from imperative

You have the parsers
Now you actually get to build the game !

# Main Server Object

Your main class should be a server
that uses a ServerSocket to listen on port 8888
for incoming socket connections
(You may have to alter your firewall settings !)

After handling a transaction with the user,
your server MUST close the connection
and listen for the next connection on port 8888

Don't panic: this is provided for you in the template

# Specifying Game Files

The names of the game config files should be passed in as command line parameters, for example:

```
java StagServer entities.dot actions.json
```

This is essential so that we can test your code (using custom game files during marking !)

Note: The starting point for the game is the first location encountered in the "entities" file

# Inheritance Hierarchy

It is essential to use a hierarchy of "Entity" classes to represent key elements of the game:

- Location: A room or place within the game
- Artefact: A physical "thing" within the game (that can be collected by the player)
- Furniture: A physical "thing", part of a location (that can NOT be collected by the player)
- Character: A creature/person involved in game
- Player: A special kind of character (the user !)

All entities will need a name and a description
Various sub-classes may need additional properties

# The Location Class

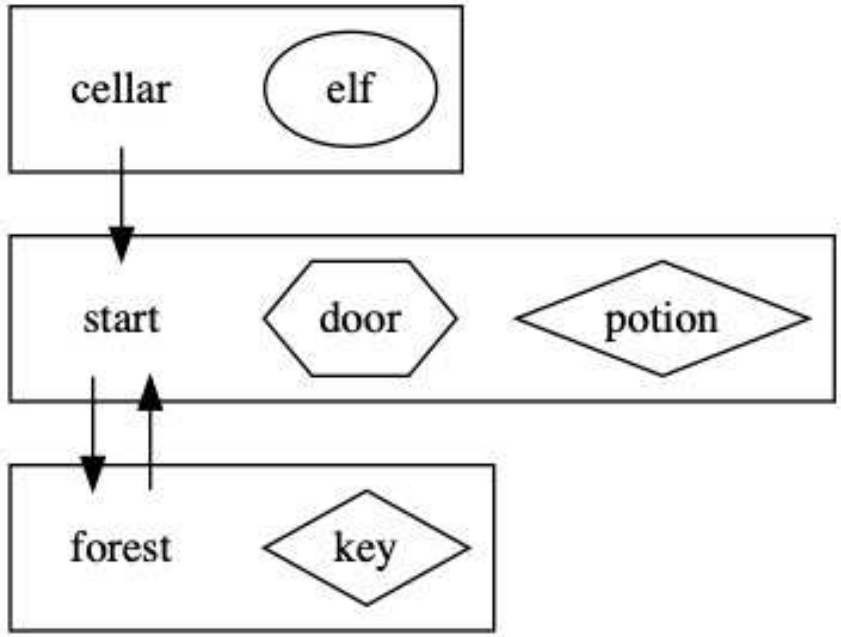
"Location" is a complex class - it should record:

- Paths to other Locations
  (note: it is possible for paths to be one-way !)
- Characters that are currently at a Location
- Artefacts that are currently present in a Location
- Furniture that belongs in a Location

# Visualising the DOT files

The big bonus of using DOT files is that there are tools for visualising them (GraphViz !)

We can SEE the structure of the "entities" file:



# Actions

Dynamic behaviour within game is represented by "Actions", each of which has following elements:

- A set of possible "trigger" words
  (ANY of which can be used to initiate an action)
- A set of "subjects" entities that are acted on
  (ALL of which be present to perform the action)
- A set of "consumed" entities that are removed
  ("eaten up" by the action)
- A set of "produced" entities that are created
  ("generated" by the action)

# Example Actions

The previous description of Actions
may be a little hard to comprehend !

Let's take a look at some examples to illustrate...

actions

# Unique Identifiers

Any names in the configuration files will be unique
So you can safely use these as unique identifiers

You won't have to deal with two things called "door"
There might be a "blue-potion" and a "red-potion"

Think of them as variable names !

# Standard Gameplay Commands

- "inventory" (or "inv" for short): lists all of the artefacts currently being carried by the player
- "get": picks up a specified artefact from current location and puts it into player's inventory
- "drop": puts down an artefact from player's inventory and places it into the current location
- "goto": moves from one location to another (if there is a path between the two)
- "look": reports entities in the current location and paths to other locations

# "Action" Gameplay Commands

Your game engine should also accept ANY of the trigger keywords from the loaded-in game actions

It must then undertake relevant additions/removals (consumption/production)

PROVIDED THAT:

ALL "Subject" entities are available to the player (in their inventory or at the current location !)

# Player Identification

Each incoming command message MUST begin with a username (to identify which player it is)

This is to allow the game to support multi-player !

A typical incoming message might take the form of:

```
Simon: open door with key
```

Be sure to make your command interpreter as flexible and robust as possible
(to deal with "varied" input from the user !)

# Health

As an extension to the basic game...
Add a "health level" feature to the Player class

Each player should start with a health level of 3
Poisons & Potions increase and decrease this number
(See the health related actions in the "Actions" file)

When player's health runs out, they return to start

If you implement these features in your game engine
Add a new "health" command keyword...
That reports back the player's current health level

# Sample Solution in action

RunStagServer

In order to connect to the server
We have also provided you with a StagClient:

RunStagClient

You won't need to alter the client code !

# Interacting with the User

You will be given a skeleton StagServer class
This includes the code to deal with network comms

You will however have to deal with user interaction
To do this you will need to use...

Buffered Readers:
cle.com/en/java/javase/11/docs/api/java.base/java/io/Buff

Buffered Writers:
cle.com/en/java/javase/11/docs/api/java.base/java/io/Buff

# Testing

A special set of custom game description files will be used to assess your game.
It is therefore essential your code is able to parse files in the same format as examples provided !

Scripts will be used to automatically test your game engine to make sure it is operating correctly.
It is therefore essential that you adhere to the gameplay input commands detailed previously !

# Going a bit further

This assignment is an opportunity to explore Java...

You might like to look at additional patterns

There are useful Java data structures you can use

A range of other interesting avenues to investigate

These kinds of things will get you the higher marks

# Fun

This assignment is supposed to fun

(At least partly ;o)

Your game engine can be used to play ANY game

(Within the constraints of the config files)

Perhaps some of you will create your own stories ?

Would be nice to share them with other students !

# Plagiarism Checking

An automated checker will be used to flag any incidences of possible plagiarism

If the markers feel that intentional plagiarism has actually taken place, marks will be deducted

In serious or extensive cases, the incident will be reported to the faculty plagiarism panel

This may result in a mark of zero for assignment, or even the entire unit (if it is a repeat offence)

# Code Quality

Code quality metrics WILL be used to assess the "quality" of your code

This will have an impact on your final mark...
So be sure to adhere to the structure and style guidelines outlined in the lectures !

As well as the quality feedback advice from OXO !!!
(Know your bad habits !)

Questions ?