Jean-Baptiste Dumat

University of Kent

CO871 – Advanced Java for Programmers

Coursework 2: Behaviour parameterisation, Streams, and GUI

Report:

This project has been coded on NetBeans using Maven. It doesn't require any external libraries. Javadocs are located in folder "target".

This software has been coded using a mix of MVC and ECS architectural patterns. The ECS, data oriented manages the whole game while the MVC manages the GUI display.

The ECS is currently one of the most powerful patterns to code video games and for example, Unity or even Unreal Engine implements it.

It basically consists of splitting the object between Entity / Component and Systems. The entity is an abstract representation of a game object. We can attach components to this entity, they are plain data and contain no logic. All the logic and update part belong to the systems.

Note that all variables hold by components are made public because a component is just data and is not supposed to know how to update itself, only the systems can do so.

ECS architecture

The sources relatives to the ECS are found in the package "**ecs"**. The Components are in a package "**component"** and all the systems can be found in various packages named "**systems"**.

EcsManager

The ECS manager is an "API" which allow to use an ECS pattern to build the game. The Class "EcsManager" encapsulates a "MultiMap" of Entity/IComponent:

- An Entity encapsulates a unique ID. Each game object is an entity.
- All components must inherit from IComponent to allow them to be stored in the map. Components are attached to an entity. They define what really is the entity.

The ECS manager uses **reflections** and **streams** to provide a lot of useful features like finding a component(s) inside of an Entity. Another important method is the forEachEntityWith. This method

applies a lambda/method to every entity which contains a specific pool of components. For example, you can decide to apply the MovementSystem logic to every entity which contains a Position and a Velocity Components and ignore the other one.

To build a new entity you need to call the “put” method (from the MultiMap). It adds a new entity to the pool and will be automatically updated by systems.

SystemManager

The Class SystemManager inherits from Runnable and must implement a run() method because it can be pass to a Thread or a ThreadPool. This method should call all the system in the order chosen by the user.

The order matters because a system might change the updates of another if it is called before, or after. If you need to change the actual SystemManager. current system loop looks like this:

```
/**
 * This method is executed in a threadPool.
 * All systems are run asynchronously.
 * Actualy the game is blocked by the input reading
 * but when the game will use an Pool of event you can
 * consider basing updates on a Clock or Framerate.
 */
public void run() {
    this.running = true;
    clock.restart();

    while (running) {
        if (clock.getDuration() >= 30) {
            // Update player and position names.
            SGuiNames.update();

            // Manage all out dialogs and write them on output stream
            SGuiDialogOutput.update();

            // Update the player's inventory.
            SGuiPlayerInventory.update();

            // Update the room's inventory.
            SGuiRoomInventory.update();

            // Update the room's player list.
            SGuiRoomPlayers.update();

            // Remove all output dialogs.
            SDialogOutputCleaner.update();

            // Disable/Enable exit buttons.
            SGuiExitBtn.update();

            // --->
            // Starts an input scan to players who got one
            SGuiPlayerInput.update();
```

The ECS is a very powerful way to optimize the creation and management of various and unique game objects. The issue due to the complexity of the inheritance and polymorphism is avoided by the fact the Components (data) and the Systems (Logic) are separated.

Multi players management

The game supports multiplayers management. A human player must own at least a component CInPlayer. All entities which contains a CInPlayer will be updated by a system SGuiPlayerInput. The inputs will be retrieved, then parsed to make a new command. You can change however you want this logic by adding/removing/modifying systems in the SystemManager's loop.

A default way to build a new Human player looks like this:

```
// Makes the player entity
ecs.put(new Entity(),
        new CName("player1"),
        new CPositionString("outside"),
        new CInventory(),
        new CWeight("0"),
        new CMaxWeight("10"),
        new CInPlayer()
);
```

Computer controlled AI

As for Human players it is also possible to make entities controlled by various AI. For this, you need to attach a component CComputerLevel. A SComputerInput is responsible for adding to the command queue the decision made by the computer. Building an AI looks like this

```
// Makes the computer entity
ecs.put(player2,
        new CName("player2"),
        new CPositionString("outside"),
        new CInventory(),
        new CWeight("0"),
        new CMaxWeight("10"),
        new CComputerLevel(CComputerLevel.Level.BRAINLESS)
);
```

If an entity owns both Components CInPlayer and CComputerLevel it means that it will be half controlled by human and computer. If you wish to change the behaviour of an AI, you just need to build new component/Systems.

Command management

Command management relies upon Command Pattern and reflection. The command pattern has been designed as an Entity.

```
Entity broker = new Entity();
Map<String, Constructor<? extends Command>> schemes = new LinkedHashMap<>();

ecs.put(broker,
        new CCommandSchemes(schemes),
        new CCommandQueue()
);

parseCommandXml(schemes);
```

A scheme is attached to a Broker entity. This Scheme is basically a map of “String/Constructor”. Command are fetched during runtime from a commands.xml file and added to the scheme by using reflection. An xml file found in config is used to define the commands.

The broker System owns a method to push commands into a Queue and a method to execute all the commands in the queue.

Logs

All logs (essentially warnings) are written in a file found in the log folder.

Game builder

The GameCore provides a way to manage the game. It can build and launches it. To build a game it uses a GameBuilder which provides two way of creating a game: Statically or by loading a csv file.

The Game source can be changed at runtime.

MVC and GUI

The GUI is managed by using MVC. The models are in fact the system of the ECS so both patterns work together.

A fxml file defines the view and is updated by the controller. The controller also acts like a notifier by queuing all users' inputs and by notifying the systems.

Commands can be added or removed. A simple way is to store them into the MenuList "Actions" by modifying the view (fxml or with some code).