MechaniQue

manual

What is MechaniQue?

What is mechaniQue?

MechaniQue is a programming language designed for the creation of interactive fiction (IF). Interactive fiction (or multiple choice question based adventure games) are fun to play but there are no really good ways to create them. Most modern programming and scripting language are way to complex. MechaniQue offers a way to develop such games quickly. Furthermore, MechaniQue is easy to learn. This manual has been written in a brief and clear way, after reading this manual you'll be able to create your own imaginary worlds. Also, MechaniQue is written in Java, which means that you only have to create your game once and it will run on any system supported by Java.

How to run..?

To invoke MechaniQue enter:

java mechanique (name of your game) (interface type) (filter)

As you already know, MechaniQue is a Java program, so you'll need a version of the Java Runtime Virtual Machine (JRVM).

Gui types : cl (commandline) gui (simple GUI version), spk (spoken texts; OS X only).

The Gui type and filter parts are optional.

Java mechanique mygame.book   - will just work fine.

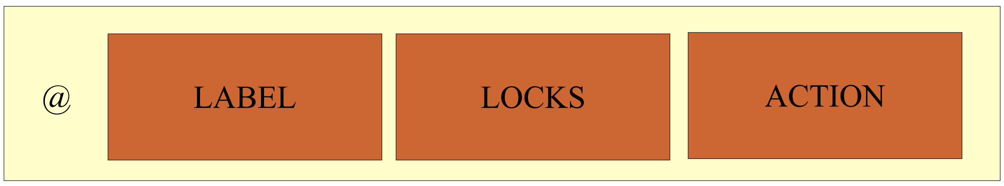
Syntax

Let's get started.

You can write your MechaniQue game using a simple editor (Windows: notepad, MacOS: textedit).

A MechaniQue program consists of code lines. Each line of code starts with a 'beginning of line' symbol: @.

Each line can contain up to three different sections:



Format:

Each line should have the following format:

```
@[X] {Y} ^Z
```

where X is the label, Y is a lock (might be more than one), and Z is the action.

Labels:

A label is just a label. It tells MechaniQue that the line has a name. Labels are useful because you can jump to different lines using label-names. The syntax for a valid label is :

```
[X]
```

where X is the label-name.

Locks

A line can be locked. If a line has been locked with one or more locks the action on the other end of the line is not performed unless the program has the appropriate key to unlock the action.

So if MechaniQue has the key “greenkey” then it can unlock a line like:

```
[mylabel] {greenkey} action..
```

A negative lock is a lock that starts with a bang, this lock opens if MechaniQue does NOT have a certain key.

```
[mylabel] {!redkey} action...
```

Actions

MechaniQue has only a few instructions. There can be only one instruction per line. A MechaniQue instruction always starts with a ^ symbol.

I will now show you all of MechaniQue's ACTIONS, substitute X with your own text:

Instruction	***Does:***	***Substitute X for:***
^:X	Prints text	Text to print on the screen
^#X	Nothing	Comment
^+X	Adds key	Key to add (without spaces or tabs)
^-X	Drops key	Key to drop
^->X	Jumps	Jumps to line starting with label X
^<-	Returns	Jumps to line after most recent jump
^?	Asks question	

The action ^? asks a question. The answer from the player is then converted to a key: all spaces are removed and the answer is prefixed with an underscore _.

There is one exception to the whole system, ending a program is done by starting a line with just :

@end

this is because as far as the interpreter is concerned 'ending' is not really an action, but more like a state of being.

Yes, this is all you need to know.

Hello world in MechaniQue looks like :

```
@^:Hello world!
@^?
@end
```

PROGRAM # 1

and a simple guessing game (asks a question, and accepts b as the correct answer; b is used as a key.):

```
@ [try] ^:What letter do I have in mind? A,B or C ?
@^?
@ {_b} ^->good
@^->try
@ [good] ^:That's correct!
@end
```

PROGRAM # 2

How to make a game

With the means described in the previous chapter you can build adventure games of the IF type (Interactive Fiction). Technically speaking, you use keys as variables. Keys are binary variables in the sense that you either have a key or not. Using your key-collection as an inventory and state-system you can do anything you need to, to develop your IF masterpiece without the hassles conventional programming languages are burdened with.

Extended Keys

To make your games more exciting the key-system has some additional features. Some keys are generated automatically and offer some dynamic features for your enjoyment.

Here is list of those special keys:

- the dice key (random key)
- the twisted key (version key)
- the historical key (absolute key)
- the rotation key (circular key)
- the path key (linear key)

These keys can be used for 'story control' adding features like growing trees / buses that pass by (circular key), moving guards (linear key), time-based secrets like Easter eggs (time key), story-twists, and randomness.

Explanation:

Name of the Keys	***Keys***	***When is one of the keys available?***
Dice	*DICE1, DICE2, DICE3, DICE4, DICE5, DICE6*	*A dice is thrown, at each ^?, the key that is available is the one that ends with the number the dice has fallen on.*
Twist	*TWIST1, TWIST2 or none*	*At the beginning of the game one of those keys is given*
Historical	*TIMEX, where X is the number of answers that have been given by the player (e.g.: TIME0, TIME261);*	*Each time the player replies a new key becomes available, all older time-keys will be dropped.*
Rotation	*ROT1,ROT2,ROT3,ROT4,ROT5,ROT6*	*After each answer the player gives, this key gets a higher number until it reaches 6, then it falls back to 1.*
Path	*PATHA,PATHB,PATHC,PATHD,PATHE,PATHF*	*Same as Rotation but goes from F to E and then from A back to B (abcdefedcbabcdef etc..)*
Occasion	*OCCX*	*Where X is the current date using MD notation: 226 means 26th of March (0=January).*

All keys can be used NEGATIVE. By using a negative lock, the lock is opened if the player does NOT have the specified key. To make a lock negative, start it with a bang: {!_a} (answer is NOT a), {!redkey} (does not have key redkey), {!TWIST1} etc..

The historical key (TIME) acts a bit different, if you have a lock in a line like TIME20, the line will be executed only if the player has interacted LESS than 20 times. As a consequence if a line has a lock like !TIME20 the actions in this line will take place only if the player has given 20 answers or more. This makes the key a bit more useful. Using the time-key you can reward a player if he performs a task within a certain time-frame.

There is one other key you need to know about. This key is known as the OR-key. An OR-key implements the OR principle. If you place an OR-lock at the end of a list of locks, mechaniQue will perform the actions if one of the locks can be unlocked.

More actions on a line

It's possible to put more than one action in a single line of code.

Actions in a multiple action line should be separated by a | symbol.

Example: @^:hello|^+key (prints 'hello' and adds key 'key').

Putting lots of actions on one line can make your program messy.

Parsing

Parsing means that the game understands normal sentences like 'go to the house' instead of just single letters like 'a' or 'b'.

Parsing in mechaniQue can be done by using a filter-file. A filter file should contain lines like this:

```
@go
@north=n
```

Each line begins with a @ symbol. The first line filters all 'go'-words from the answers. The second line contains a = and therefore replaces north for n. Imagine a player answers : “go north”, using this filter this answer will result in key: _n. However “north” will also result in _n. So you can use the same key-system, the parsing is just done by the filter file. To start a game with a certain filter:

```
java -jar mechanique.jar mygame.book cl myfilter.txt
```

Errors & Limitations

There are two types of errors that may arise when running a MechaniQue program; unknown instruction or anonymous error. An anonymous error means that the program just could not be executed any further. It provides a line-number but no further information on the exact cause. The unknown instruction error means that at the specified line there is an action which is not part of the current MechaniQue implementation.

There are some limits you need to know about: 1. a single line can contain a maximum of 40 locks. 2. Your key-collection can store up to 800 keys, however 10 keys are reserved for system usage. If you drop unused keys you keep the inventory clean and you will not run out of key slots.

User Commands

After invoking a mechaniQue adventure there are several special commands:

#about	: shows about information
#st	: shows memory usage

History

MechaniQue was originally written in Javascript, however, because I wanted to practise with Java, I decided to write it again in Java. I also redesigned the language, making it more focused on IF. I designed the key-system to replace the conventional expression system and simplified and reduced the instruction set.

Examples

As you have seen MechaniQue is a very easy-to-learn language. If you still think you need some practice, take a look at the examples

(.book files).