Dream

WGan

# Preface

This article describes design of quest game engine and language defining the game. It does not describe game itself specially it does not describe graphics, animation, special effects, however some technical hints can be found here. Prototype of the game is written in python to illustrate algorithms and design of the engine. Prototype is fully playable.

# Interface

To check the engine a simple interface should be provided to allow functional testing and test play. We decide to provide text interface, as in typical RPG games, focusing on plot and leaving the rest to player imagination. Games can be used as interactive books – and in this form can be used as trainings and tests in various teaching activities.

Games could be also designed by students as projects on various subjects starting from history, through contemporary languages ending on programming by expanding game engine by pictures or animations.

# Definitions

Words and phrases describing elements of game engine are described here.

## Item

Item is a game element which can be used in orders when it is available. Items can be assigned to locations, carried by the player or crafted.

## Location

Location is a place where items and/or players can be located. Location can be – part of forest, room, cabinet, backpack. Locations are more or less independent, however typical location contains ways to other locations. Change of location can be performed by the order. Locations have no names, however internally are identified by unique codes. Typically locations have description which is printed when player enters location. Also some states of game can be described as virtual location. Example of virtual location can be fight or discussion with creature.

## Command

Order describes action performed by the user. System distinguishes two types of orders:

1. Issued command – command issued by the player or by creature (bot) describes action which should be performed. Issued command contain action name and may contain names of items.
2. Available command – command attached to player, item, location or realm describing command which can be executed. Contain condition defining which property values must be reached to execute command and change which described how to modify properties and set of items at location, item or player.

## Property

Named numeric value attached to item, location, player or any other element of the game. It can be understood as property of object in OOP meaning. Properties are checked and changed when processing commands

## Player

Player is a character having its own properties and inventory, travelling through locations and performing actions by executing commands.

## Realm

Entity defining common set of available orders and properties which can be assigned to location. Simplifies definition of location allowing definition of common action and common behaviour. Realm can also define operation which is performed at each game step simulating passing time.

## Creature

Creature is automatic player whose behaviour is defined by the program written for this creature. Program is defined as set of orders which creature try to execute. No more than one order is executed successfully in single game step.

# The game

The Game is played in turns. Each turn contain (in the order):

1. printing description of situation or situation change after previous
2. execution of all actions bound to the *location* (including *realm* actions);
3. execution of all actions bound to *creatures* which are in this *location*;
4. execution of *player* command (when *command* cannot be executed *player* is asked for next *command* till it is successfully executed);
5. printing result of command

Description of location and items are printed as they are in database. Commands provided by players are simplified English sentences having construction:

**command [item] [with item [, item]\*]**

where **command** is the name of action, typically described with one word, however more complex phrases are acceptable. **Item** is the name of an item, creature or player and can be described with more words. In orders full name of order and full name if items.

When order was successfully executed, its result is described as change of item, location or player. When order cannot be executed – message “I do not know how to …” is printed.

# Definition of potential command

Potential command is a command which can be executed. Potential commands can be executed by the player, and defines command by specifying:

* word or chunk of words defining the command
* parameters which specifies what should be used to execute command (which item/location/creature stats or location should be modified or used when executing command
* set of conditions. Each condition is set of property limits. Command is executed when condition in set is satisfied.
* Set of operations. Each operation can be:
  + Modification of any property in player, location or mentioned item. In modification +, -, \* and / can be used. Also special internal property ‘rand’ can be used as random number.
  + Destruction of item in player inventory or location
  + Creation of item in player inventory or location
  + Printing text or long description of mentioned item or location
  + Printing player or creature inventory
  + Changing location
  + Exiting the game

Example of command:

{“craft sword”} WITH tool WHEN tool.weight > 1000; tool.hammer > 5000; player HAS “bronze bar”; player.crafting > 50 THEN player LOST “bronze bar”; player EARNS “bronze sword”; PRINT “You crafted bronze sword. It is not very sharp.”; player.crafting += 5

The command look like normal sentence and it is easy to understand. However lets decompose it:

* “craft sword” is the beginning of the use command. The command will be executed when user types “craft sword with hammer”. Notice that this text is written in quotations because it contain more than one word. When more commands match the same. If more command text variants are possible – place semicolon separated
* WITH – keyword separating command name with parameters. When no parameters are needed, this keyword is also not needed.
* Tool – item used in command when command need item to be used as a tool or source.
* WHEN – keyword. Separates declaration of command and conditions.
* tool.weight > 1000; tool.hammer > 5000; player HAVE “bronze bar”; player.crafting > 50 – list of conditions separated by semicolon and surrounded by brackets. All conditions must be true to execute command. In condition, expression can be used, however only simple expression containing basic four operations, random number constants and named properties of player, location and item passed as parameter. It would be good to have access to any item in location or player by naming it (like: player.bronze\_sword.sharpness). To access an item – any of its name can be used. If name is ambiguous – operation try to execute with each of them – first successful is used.
* THEN – keyword separating condition and list of operations
* player LOST “bronze bar”; player EARNS “bronze sword”; PRINT “You crafted bronze sword. It is not very sharp.”; player.crafting += 5 – list of orders to execute when all conditions are met.

Notice that there is no ‘ELSE’ part. If different behaviour of order is defined – the same order can be defined once again with different condition and different statements list. Executed is first command in the list which match command name and.

To highlight keywords, all are written in capital letters. It also separate them from game defined commands. Keywords are sometimes at the beginning of command (like PRINT “what” “to” “print”) or in the middle (like player EARNS sword). When compiling or interpreting commands, in each command engine looks for keyword and then interprets words around as parameters.

# First milestone - architecture

First step of programming – first goal is the architecture with proof of concept. Main data and processing classes need to be designed, prepared, tested and combined to archive simplest game ever: One location with one player and only one order available: “look” returning description of the location.

# Command

Potential command is represented by the expression tree or list. Each node makes its own processing, and returns with False – when it decides that the command was not executed, or True – when the command was successfully executed. Node, depending on its type can:

* execute predefined action – like printing description on console , exiting game, changing current location, creating/morphing/transforming item etc.
* execute list of actions as long as the returns true or false (or-like and and-like list of commands)
* guard needed prerequisite based on available stats. Stats can be read as total, maximal or pointed. Here also trees or expressions are used. If guard fails – it immediately return False. If it pass, it returns result of processing of next node.