**Procedural Content Generation using   
Recurrent Neural Network (LSTM) in Lode Runner**

**Author**

**Ivanov Silviu-Gabriel**

**Gr.234**

**2018**

**Problem statement**

During the years procedural content generation was used in order to generate different types of content. This document tries to highlight the combination of machine learning with procedural content generation. An important field of machine learning that maps over our problem are the recurrent neural networks, this approach presenting several benefits in the case of linear level generation. In this paper the RNN is used to generate Lode Runner levels in order to be able to generate an infinite amount of levels from the base levels of the game.

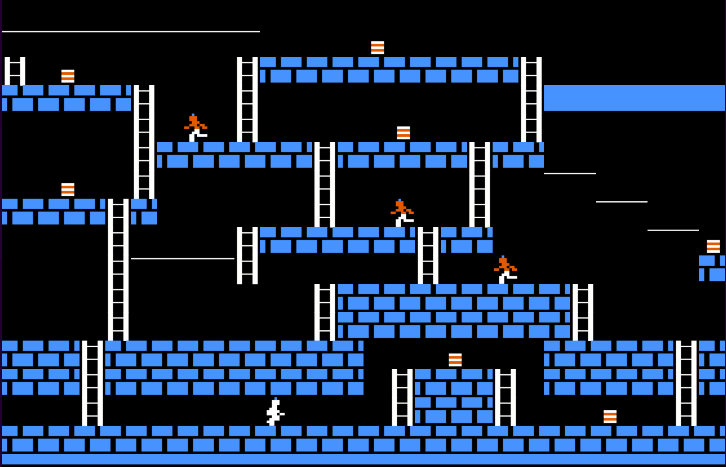
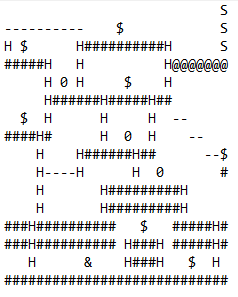
**Problem analysis**

*Research method*

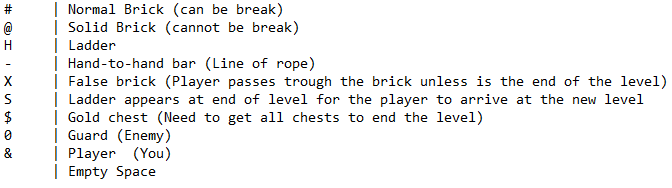
To be able to generate an environment in this project we need a target game with original levels can be used as a database. The target game for which levels will be generated is the **Lode Runner**. The evaluation a training will be written in python using the Keras library. In this project we are using a JavaScript implementation of Lode Runner. *(*[*https://github.com/SimonHung/LodeRunner\_TotalRecall.git*](https://github.com/SimonHung/LodeRunner_TotalRecall.git)*)*

*Level representation*

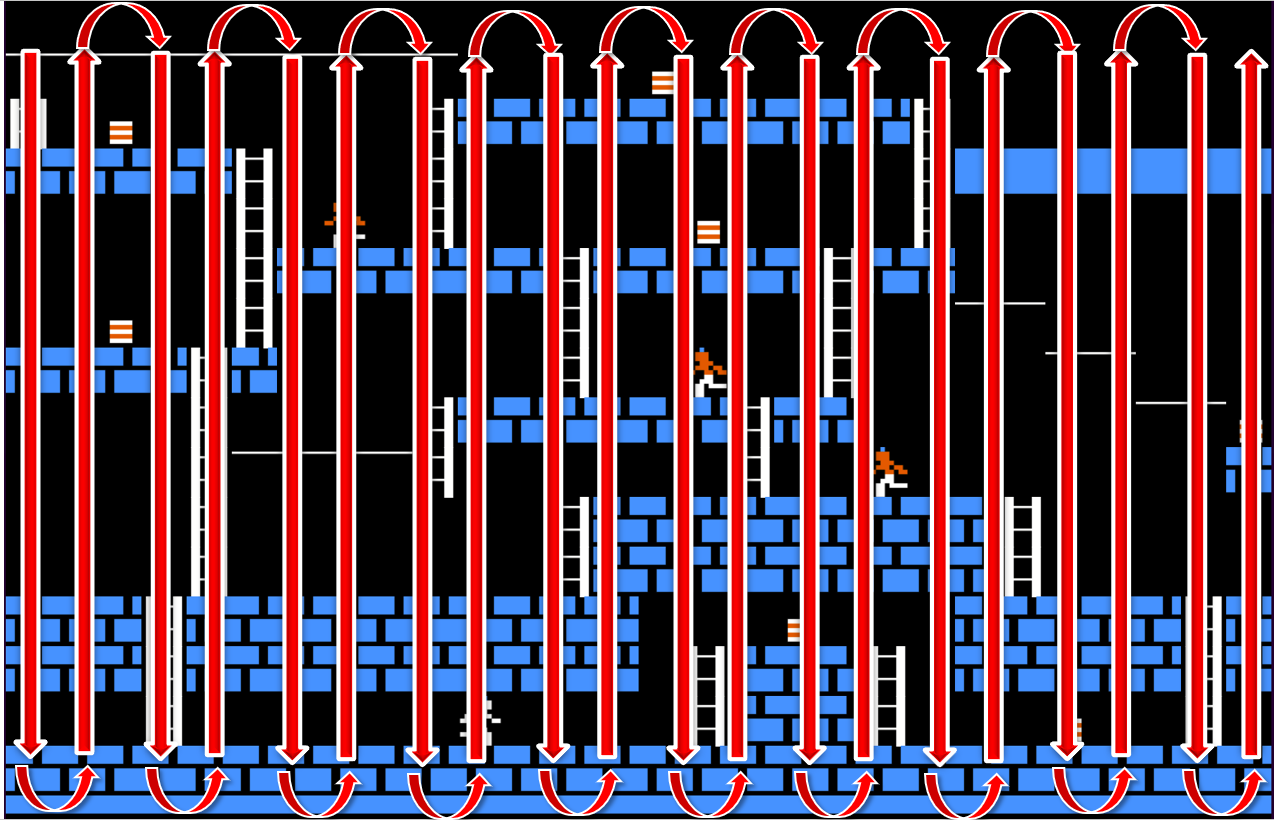
The level of the game consists of several blocks, ladders, enemies, gold chests, hand-to-hand bars. Each level of the game will be represented as a text file, where each element is encoded in a token.



The dictionary of the level:



The level will be processed in a “Snaking way” on columns, this will transform the input in one hot array of tokens to be forwarded to the LSTM.

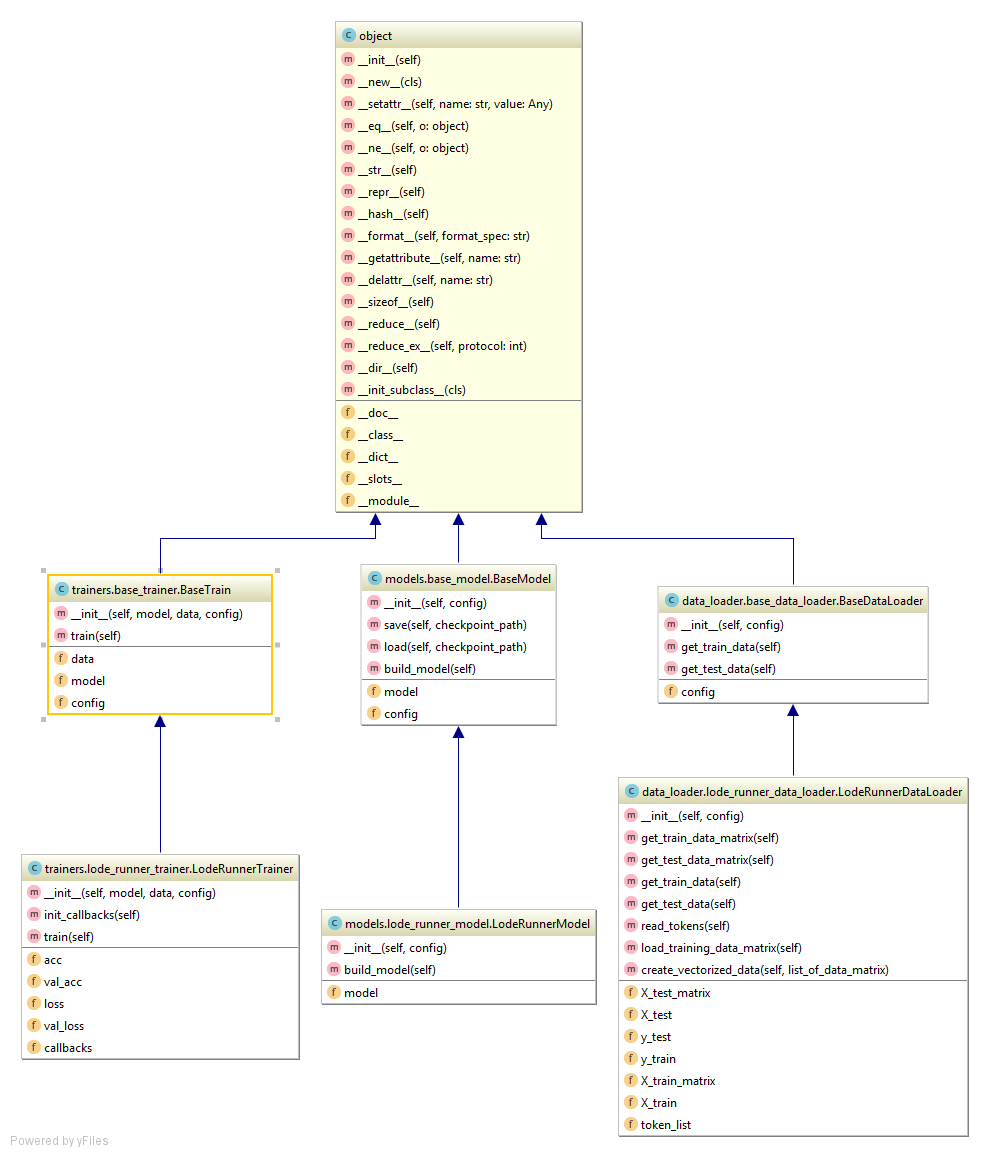


Lode Runner consists of 150 levels, that will be used for the training and evaluation. 100 Levels will be used for training and 50 for evaluation.

Each token will be later transformed in a vector with the size of dictionary of tokens, where every value is 0 except the value corresponding to the token.

We expect after the training to be able to introduce a vector that corresponds to a token and get a vector of probabilities for each token in the dictionary.

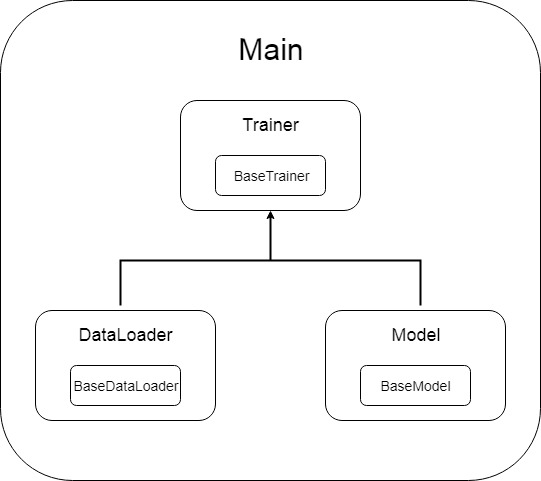
**Design and Architecture**

****

DataLoader – class that load the data into memory and prepare it for the process of training.

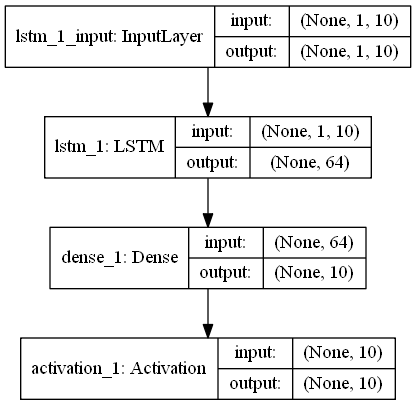
Model – class that specify the architecture of the neural network.

Trainer – class that train the model with the data from the DataLoader.



**Model**

The model is created using a sequential model in Keras with one LSTM layer, one Dense layer and an Activation layer. The purpose of the Dense layer is to be able to get the output add up to 1 (list of probabilities for each token in the dictionary). The loss function used is a “categorical\_crossentropy” used for “multi class log class”.



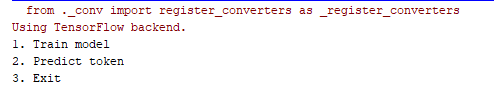
**User Manual**

The project will have a console with 3 options:

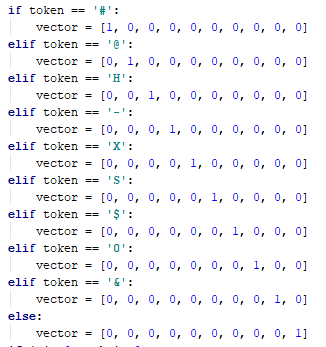
1 - for training the model based on the config file *lstm\_config. json*

2 - for introducing a token and get the probability vector for all the tokens

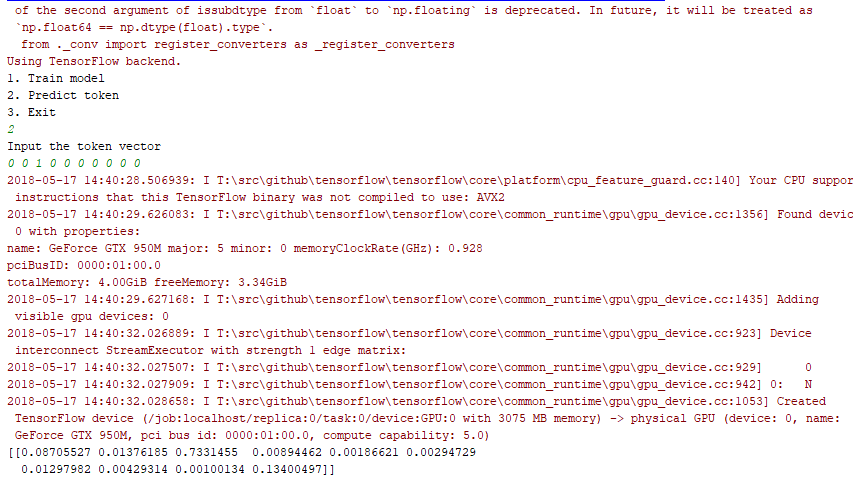
3 - exit the application



In order to introduce a token, we have to represent it using a vector as follows:



For example, we will use a token for a ladder [0, 0, 1, 0, 0, 0, 0, 0, 0, 0].



We can see that the result vector is

[0.08705527 0.01376185 0.7331455 0.00894462 0.00186621 0.00294729 0.01297982 0.00429314 0.00100134 0.13400497]

meaning that the token with the highest probability is also a ladder.