**Knights and Knave**

Author - Vatsal Paresh Gala

Email - vatsalpgala@gmail.com

Date- 26th April 16, 2019

**Overview**

This document explains the working of the Knights-and-Knave puzzle generator. On an abstract level, the puzzle generator makes use of Genetic Algorithm to come up with puzzles (represented in Boolean equation) from a randomly selected solution. The boolean equation consists of blocks (where each block represents a true claim for each inhabitant irrespective of it being a knave or knight).The Boolean equation is then translated into verbal format through some fixed predefined rules. For simplicity purposes complex sentence formations are avoided and simple sentences are used. The document contains 4 sections: Section 1 talks about the terminologies, notations and assumptions used in the project. Section 2 and 3 talk about the important methods / functions / data structures used in the project and about algorithms and processes used in the project. The last section describes how to run the puzzle generator on a local machine.

**Important Functions and Data Structures**

1. Function - *create\_boolean\_question*

Main function that creates boolean equation (a logical representation of question). Runs the genetic algorithm and converts the logically correct bool equation according to the claims of knights and knaves(i.e. Knaves always lie, thus their bool claims are negated).

1. Function - *equation\_to\_sent*

converts logical boolean equation to verbal representation. Follows certain set of rules that translate bool to natural language.

1. Class - *Genome*

class object that hold individual Genome, and its functions. Each member of population is a <Genome obj>. The genome basically contains boolean equation that might be a potential questin to the assumed knights and knave combination.

* 1. Function - *calculate\_fitness*

based on the truth values<knight and knave combination> for variables in bool equation as compared to the desired truth value of the solution, this function gives score to each Genome obj, paradoxical equations, equations with multiple and no solution are given a comparatively lower score. Each truth value combination that satisfies the <Genom obj>e equation is considered to be contributing to its fitness score. Hamming distance of 0 between the truth value combination and assumed answer<knight and knave combination> is given the highest score (3) and hamming distance of 3(for 3 inhabitants) is give a -3 score.

* 1. Function - *crossover*

Uses one-point crossover method within a block. Randomly selects if a block must undergo crossover or not. Each genome is randomly chosen with another randomly chosen Genome from <selected> list. Each crossover operation yields 2 child Genomes.

* 1. Function - *mutate*

Uses single point mutation over entire Equation and ensures paradoxes are not created while mutating.

1. Function - *make\_random\_block*

For 3 inhabitants, randomly choses 3 variables from list of ['A', 'B', 'C', '1', '1'], where '1' is a placeholder for consistency purposes. Then it randomly negates variables and selects among operators ['|', '&']. This forms a block that represents a claim by a single inhabitant in a question.

1. Function - evaluate

Evaluates a boolean equation by removing placeholders, substituting truth values for variables and operating based on logical operators. Returns a True or False value.

1. Function - *SUS, RWS*

SUS - Stochastic Universal Sampling, RWS - Roulette Wheel Selection. Sampling and selection methods for crossover. (Wikipedia offers a more in depth explanation of these algorithms)

1. Data Structure - *truth\_values*

The project works on certain assumption for adequate representation of puzzle in terms of boolean equation. Knights: True, Knave: False, 'A' by default represents, 1st inhabitant as Knight, '-A' by default represents 1st inhabitant as Knave.

1. Data Structure - population

It is a list of all genomes.

1. ANSWER - variable containing Selected Genome obj after Genetic Algorithm runs
2. QUESTION - variable containing the final Genome obj with actual actial question represented as boolean equation.

**Algorithm**

1. Generate button<html> create an AJAX url request to flask\_server

2. Flask server runs internal function.

3. Call *generate\_boolean\_equation* function.

3.1. set global variables/ control parameters

3.2. Randomly assign Knights and Knave value<theory>

3.3. Generate a starting population

3.4. Start loop

3.4.1. Calculate fitness of each individual in population˜

3.4.2. Pick elites based on the fitness score

3.4.3. Check for solution in the elites, If found, exit loop

4.3.5. If Answer not found, select population for breeding

4.3.6. Crossover existing population to create child

4.3.7. Randomly mutate

4.3.8. Elite, children and new randomly generate individual form the next generation

4.3.9. Repeat steps 3.4.1 to 3.4.8

3.5. Based on knights and knave value negate the boolean equation for knaves

3.6. Return new equation as QUESTION<Genome obj>

4. Randomly assign names to variables

5. Call *generate\_question* function

5.1. Compile The question using standard, introduction, and names

5.2. for each block\* call equation\_to\_sent function

5.2.1. Remove '1'(placeholders)

5.2.2. Based on variables left in the block, follow separate set of rules to compile sentence.

5.2.3. Return string value consisting of verbal representation of boolean equation

6. Wrap answer in JSON format and return it back to the html page.

7. Display the result

**Running the Code:**

Requirements:

1. python 3.5+ (preferably) but should work for any python 3+ version. Installation : https://www.python.org/downloads/
2. Flask library (python library): Installation from here. (if required) : http://flask.pocoo.org

Steps:

1. Open terminal / cmd here.
2. Navigate to ‘Knights and Knave/Prototype’ folder.
3. Run command ‘python flask\_server.py’ (Keep the cmd / terminal open during the entire process).
4. Open Browser of choice.
5. Visit url : localhost:5000/knights-and-knaves
6. Click on generate button for new randomly generated puzzles.

**Limitations:**

Since, the system generates puzzles using Genetic Algorithm, sometimes it may take longer time than others to reach to a conclusive solution and thus each run time may differ from each other minutely (also depend on the processing capacity of the hardware the system runs on).

The application is thoroughly tested and is unlikely to give unexpected or false results. But in case of any errors, feel free to contact the author for any rectifications.