*Stone Age hunter-gatherer experiment V 1.0*

*Originally submitted as "Programming for Cognitive Science Assignment 3" at University College London, 6 March 2017*

**Experiment overview:**

The present experiment investigates how people make choices in exploration vs. exploitation dilemmas of increasing complexity and with different underlying levels of variance. Specifically, it is hoped that through cognitive modelling enabled by the extensive amount of data gathered throughout this task, it is possible to better approximate how people make decisions; when their decision-making is optimal in light of imperfect information, and when it is not. There are three conditions that differ in terms of the parameters assigned to probability functions, and by how much a previously exploited resource depreciates on the next trial. It is hypothesized that with increasing variance and complexity of the task, participants perform worse, however they can use visual cues from the resource patches as well as their previous experience in the task to infer the underlying payoffs.

**Description of procedure & Experimenter's manual:**

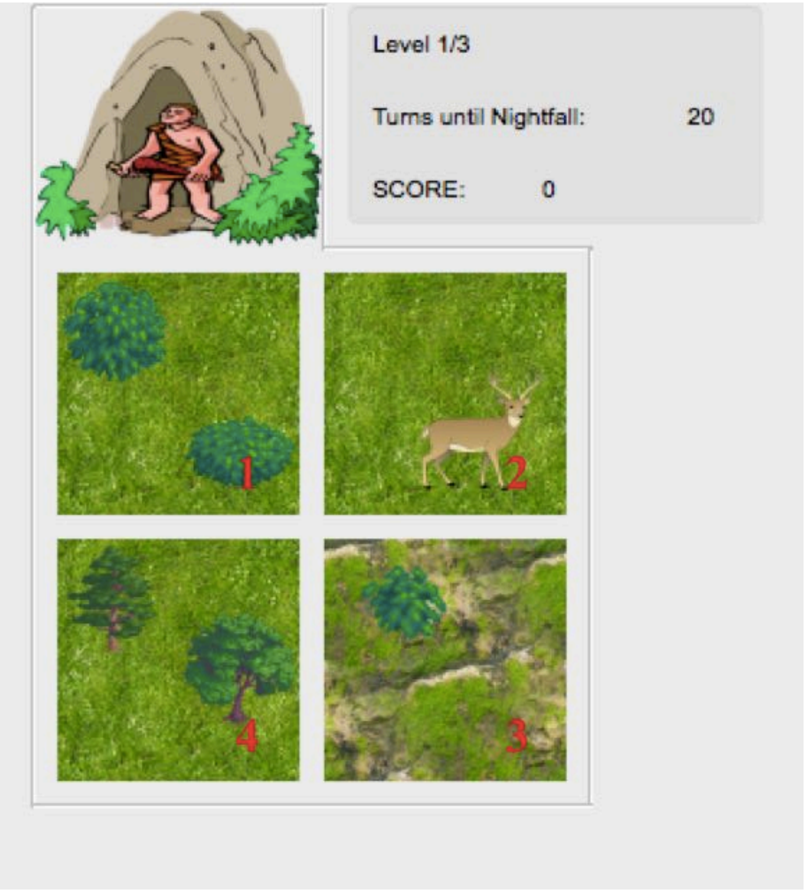
Participants will undergo the entire experiment on the computer; this includes the consent and debriefing pages, as well as the entire experimental output. The experiment will not go through without agreeing to consent and correctly filling out the demographics page, it is of course anonymous.
Every participant will need his unique participant ID to fill in on the demographics page. This should be allocated by the experimenter well in advance. It must be unique for every participant, but should be randomised and independent of e.g. student IDs.

Because the task itself is rather challenging, participants are best rewarded for their achieved score with some money; how this conversion should go through remains to be determined, but it should be substantial enough that it is a real incentive (e.g. double the signup reward).

It is very important that they understand the experiment properly. It is in principle easy understand, but difficult to master; they will move their gatherer around using the arrow keys and may choose to exploit any resource patch it is currently located on. Both moving and exploiting cost movement, but only exploiting can result in payoff. When all moves are wasted, they must be back at their cave, or will be eaten by predators and lose their score of that level.

While normally the expected payoff per resource is always positive, variance is large and there are many catches that make learning difficult. Nonetheless, with the right incentive they may still learn. Some features of the patches are predictive of payoff and risk, but others are not. When there is rocky grass, payoff can be more negative, whereas game will result in large payoffs, but also a set chance of being eaten. There are some more detailed explanations in the code, in particular the specs.py file that contains the payoff parameters.

If the participant enters any feedback, this is also logged in a separate .txt file, so that feedback given does not clutter up the results file. Both files are saved in the folder where the programme is.

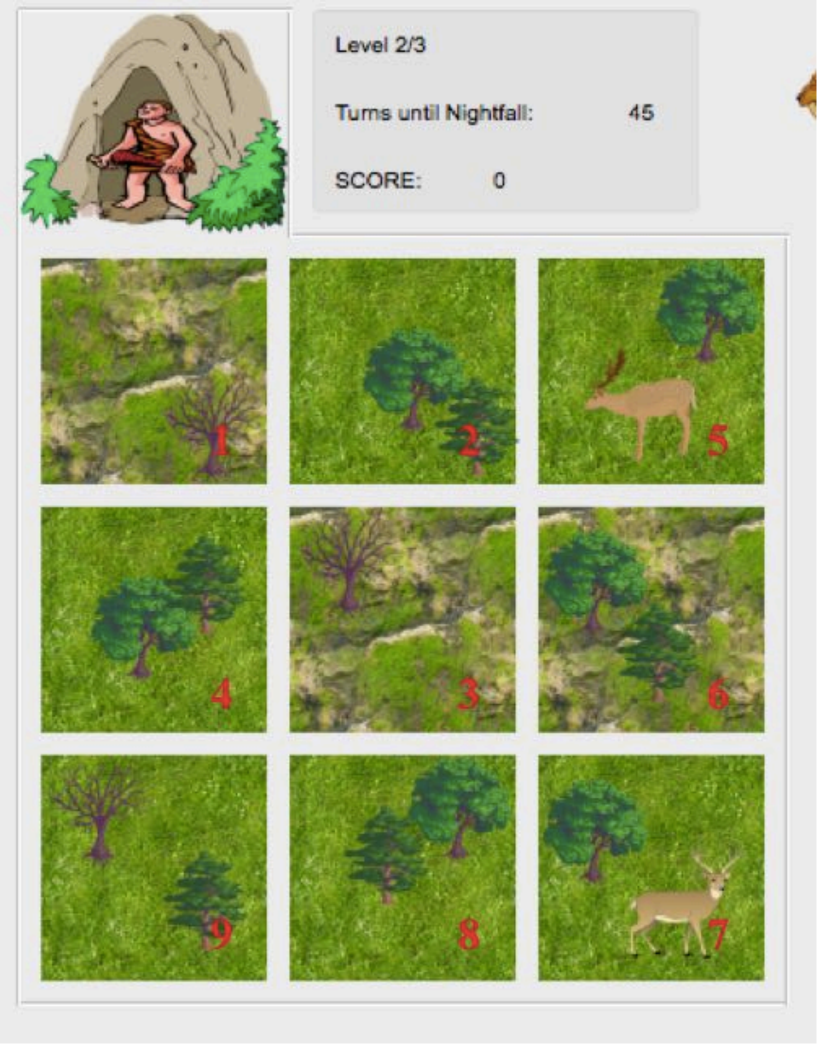


Level 1/3

Turns until Nightfall: 20

SCORE: 0

Figure1. Arrangement and naming of resource patches in level 1.

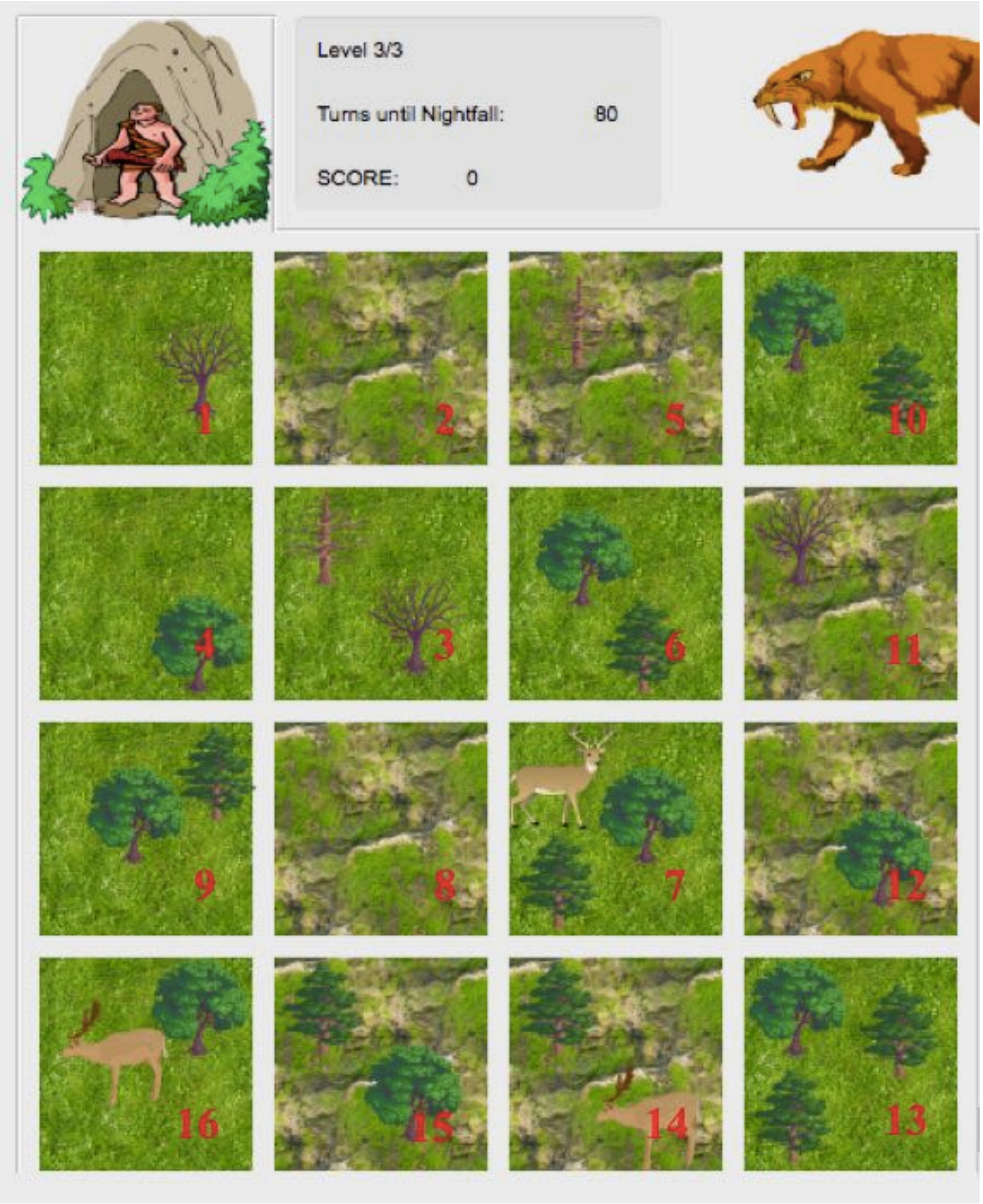


Level 2/3

Turns until Nightfall: 45

SCORE: 0

Figure 2. Arrangement and naming of resource patches in level 2.



Level 3/3

Turns until Nightfall: 80

SCORE: 0

Figure 3. Arrangement and naming of resource patches in level 3.

Patches are numbered clockwise on level 1, and this count is continued for every appended layer of patches after that, so that a certain patch number will refer to an identical position in terms of x and y coordinates, independent of level (Figures 1- 3).

Every patch generated payoff when exploited by drawing from a probability function. The parameters and type of this function differ both for the condition a participant is assigned to, as well as the properties of the patch.

Across all levels, there are two types of patches: grass patches and grassy cliffs (the patterns are in reality pictures, while everything else is clipart). Grass patches have a standard Gaussian distribution with a set mean payoff and standard deviation, whereas grassy cliff patches have a multimodal distribution that is much harder to learn; i.e. payoff is randomly drawn from two different distributions, but this is very hard to infer.

Trees & bushes: are not predictive of reward in this implementation.

Game animals: If a deer is present on a patch, the mean payoff is increased by a flat 10, but every time it is exploited there is a 10% chance that the gatherer is eaten by the sabre tooth tiger.

If the gatherer is eaten by the sabre-tooth tiger, the score obtained thus fair is reset to zero. Thus, being eaten at level 3 will result in a total score of zero.

**Program highlights**:

This programme should be noted for increasing the complexity of the scenarios from level to level, as well as for offering a great many opportunities for expansion and further testing. For instance, different parameters for the probability functions and higher allowed movement per level can be explored, different types of patches added etc. In any case, the goal would be to explore an optimal strategy that would result in maximised payoff both in case the model parameters are known, and when they are not, and to compare this strategy to how people actually behave when exposed to the task.

The code is highly concise and avoids repetition wherever possible (often executing or evaluation concatenated strings) and highly readable with comments and easy to understand variable names.

Because the probability parameters are read from an external file, they can be easily modified, e.g. by multiplying all with a set value. In the current programme, these are expressed as difficulty variables that differ per condition.

Great care has been taken to log all data that would be necessary for extensive cognitive modelling, thus every action that participants make is logged and written to the results file.