Institut Polytechnique de Paris

TPT-DATAAI961

Self-organising Multi-agent Systems

Self-organising Contraption for Automatic Task-distribution

Yuyan Zhao

February 9, 2023

Contents

1 Introduction

The task-distribution problem refers to distribute a set of tasks of different types among a set of agents in the system. This problem is of important practical sense in many fields including operations research, computer science, and engineering. It can help optimize various objectives such as maximizing overall productivity, minimizing completion time, balancing workloads, and ensuring fair distribution of tasks. The process of task-distribution can be self-organised and decentralised, which means that there is no single centralised entity manipulating directly the distribution.

This project aims to simulate a decentralised task distribution via a 2D contraption. An agent-based model (ABM) was built from scratch with NetLogo to address this problem, including different kinds of agents, interaction rules amongst agents and with the environment. First, this report illustrates the design of the model. Then the implementation on NetLogo is explained, as well as the user manual. Finally, an evaluation is made to analyse the results of the simulation.

2 Model Design and Implementation

The main principle to design the model is that the tasks can be modelled as objects provided from one or several sources. Therefore, the tasks are represented as balls of different colours, and the targets are represented as buckets with the corresponding colour in the project. In order to make balls be automatically distributed to the targeted buckets, the physical laws for balls and self-assembly laws for the contraption are defined, as well as some obstacles.

2.1 Environment and Turtles

The first step is to initialise the environment and create turtles with different types in the setup function. To begin with, several global variables have been set, such as the world size, the list of the colours for balls and for obstacles. As the buckets are not required to be moved, they can be defined with patches in the corresponding colour of the balls.

There are three kinds of turtles defined in the project: balls, sloping obstacles (slashes) and horizontal obstacles. The ball will be generated at the top with random x-coordinates. Some private attributes of the ball have also been defined to store information, such as the detected results of obstacles and the acceleration. The slash consists a set of squares with the same identifier and the increasing coordinates along the left or right direction. To make it easier to manipulate them in the following operations, these squares are linked and tied as an entirety. Similarly, horizontal obstacles can also be created in this process.

2.2 Physical Laws for Balls

There are two physical laws stated for balls. The first physical law is free fall, which allows the ball to drop when it not hit an obstacle. The second is rolling, which let the ball roll on sloping or horizontal obstacles with the acceleration. Before applying the laws, the ball needs to do a series of checks, including to check if it met the target bucket, if it hit an

obstacle and if it can continue to move. For example, when the ball falls to the ground, or when it has no acceleration on a horizontal obstacle, the ball will be asked to die.

2.3 Self-assembly Laws for Contraption

In previous work, a static task-distribution contraption has been completed. To achieve a self-organising contraption, two sorts of self-assembly laws have been specified.

The first kind of self-assembly laws is to move obstacles. Balls can observe the nearest obstacles and buckets, then they aim at the bucket and send messages to the nearest obstacle to make it move.

Algorithm 1 move-obstacle

```
ask ball to observe the nearest bucket
if ball is on the right of the bucket then
  chose the nearest forward slash
  if ball is not on it then
    if slash-length < distance-to-bucket then
      move slash-right-end to ball
      chose the nearest obstacle
      if slash-length + obstacle-length < distance-to-bucket then
        move obstacle-right-end to slash-left-end
      else
        move obstacle-left-end to bucket-right-end
      end if
    else
      move slash-left-end to bucket-right-end
    end if
  end if
else if ball is on the left of the bucket then
  chose the nearest backward slash
  if ball is not on it then
    if slash-length < distance-to-bucket then
      move slash-left-end to ball
      chose the nearest obstacle
      if slash-length + obstacle-length < distance-to-bucket then
        move obstacle-left-end to slash-right-end
      else
        move obstacle-right-end to bucket-left-end
      end if
    else
      move slash-right-end to bucket-left-end
    end if
  end if
end if
```

Based on this algorithm, the left and right ends of the obstacles are taken into concern to avoid possible position conflicts during the movement. To further optimise the positions of balls, buckets and obstacles, the second kind of self-assembly laws is defined, such as removing the obstacle that covers the buckets, keeping obstacles from being moved out of view and preventing the ball from falling directly onto a horizontal obstacle.

3 User Manual

In this section, a user manual is written to provides users with information and instructions on how to operate the NetLogo program for this project. It gives a step-by-step guide on how to set up and start using it.

3.1 Adjustable Parameters

We start with the adjustable parameters settings. There are 5 adjustable parameters, the values of which can be modified by dragging the slider. The first parameter "n-make" refers to the number of balls created by per button click. The next two parameters are "n-slash" and "len-slash", denoting respectively the number of slashes for each direction and the length. "Slashes" are sloping obstacles, indicated in the red color. Similarly, the last two parameters are for horizontal obstacles that in grey.

3.2 Function Buttons

After setting the parameters, we can click the set-up button to initialise the program. You can see that the 2D contraption is constructed according to the parameters set. Then we can click the go button to start a simulation to distribute the balls generated to the buckets with corresponding colour. If more balls need to be created, we can modify the parameter "n-ball" again and click the make-ball button.

3.3 Results Monitors

There are 4 monitors and a plot in the bottom half. The monitor "n-balls" is the total number of balls generated. "n-distributed" presents the count of balls that have been distributed to the correct bucket, while the "n-deadballs" denotes those which failed. The successful rate is the result of "n-distributed" divided by "n-balls". And the plot shows the variation of the distribution successful rate with timesteps.

4 Results and Evaluation

As previously explained in the user manual, the evaluation metrics are "n-balls", "n-distributed", "n-deadballs", the successful rate and its plot by timesteps. In the following figures we can see that different parameter values can lead to different results. In order to optimise the distribution outcome, it is essential to carefully adjust the parameters.

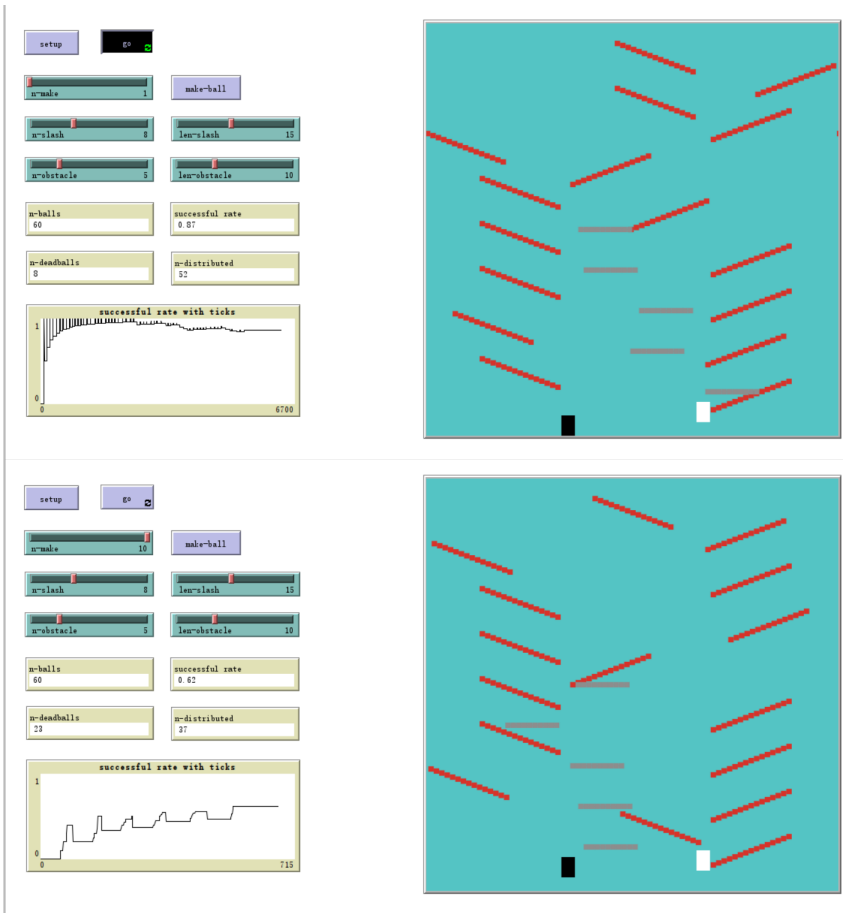

Figure 1: Results with 1 ball or 10 balls created per button click

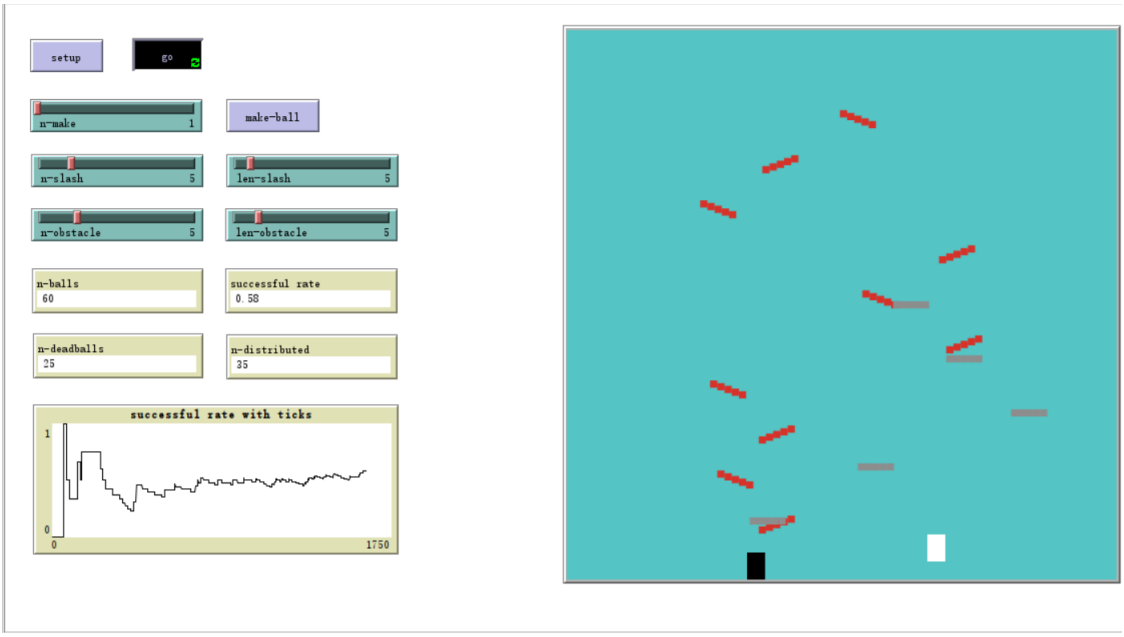

Figure 2: Results with small number and length of obstacles

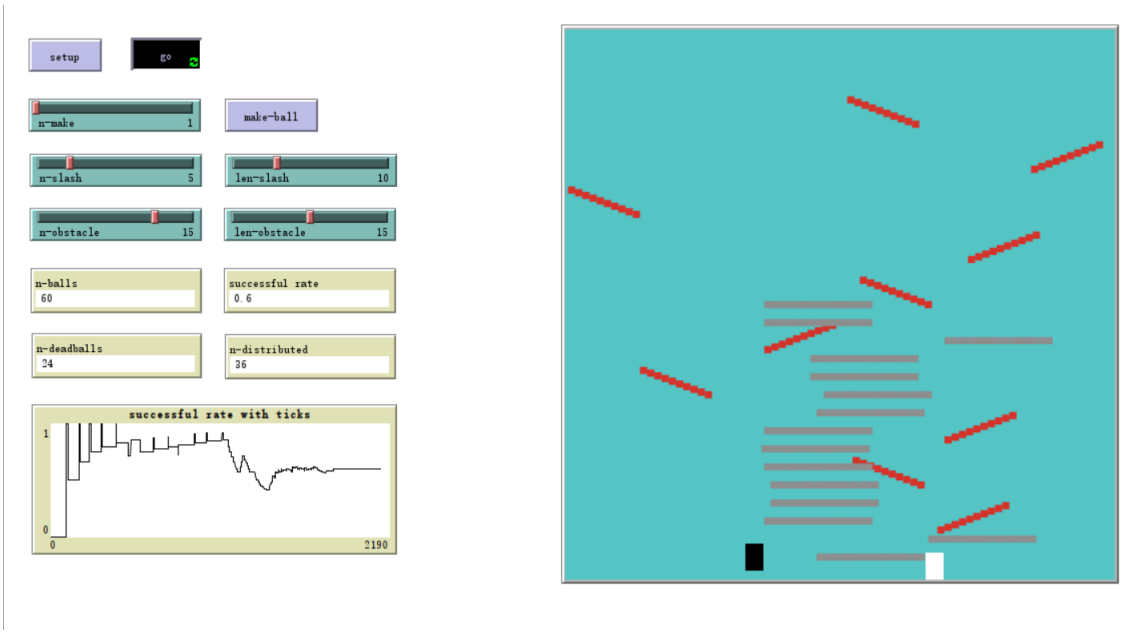

Figure 3: Results with many horizontal obstacles

5 Conclusions

From the Figure 1 we can see, the more balls to be distributed in the same time, the lower the successful rate it will be. According to the Figure 2 and the Figure 3, if the number and length of obstacles is too small, the contraption will be incompetent to distribute balls with long distance. However, if this number is too big, it will be difficult to handle the conflicts between obstacles. Besides, if there are too many horizontal obstacles, the ball is likely to die on them without the acceleration.