Genetic programming for building voxel structures with turtle robots

Iacobescu Tudor

2021

Chapter 1

Introduction

This thesis will explore the construction of three-dimensional voxel structures using a "turtle"-type virtual robot. This robot is programmable, with the ability to interact with its surroundings by means of a set of elementary functions. These functions allow it to move, place and destroy blocks, and detect adjacent blocks for the purpose of making decisions.

This project initially took form based on an existing concept: a modification for Minecraft that adds such a robot to the game, which is programmable using Lua. Minecraft is a sandbox video game where players can build structures using 3D blocks aligned to a grid. The modification, *ComputerCraft*, adds simple Lua-programmable computers to the game. Among them is a mobile computer, called a *turtle*, that can use a special library to interact with its environment.

The idea that the thesis is built around is using genetic programming to create, given a target voxel structure, a program that can construct it. When run by a turtle with sufficient materials, this program should allow it to reproduce the initial target as accurately as possible.

Chapter 2

Practical context

In this chapter, I will outline the precise problem that needs to be solved, as well as the constraints of the work environment.

2.1 The turtle

The turtle is a robot that takes up one space in the voxel grid. Its state is described using a triplet of integer coordinates, as well as a direction aligned to one of the horizontal axes.

For the precise abilities of the robot, I will take inspiration from ComputerCraft, the previously mentioned modification. From the library of commands available for turtle control in ComputerCraft, I will utilize only the following:

- The commands `forward`, `back`, `up` and `down` allow the robot to move to an adjacent space. If said space is taken up by a block, the robot will remain in place. These commands will return a boolean value, equal to the success of the movement.
- The commands `turnLeft` and `turnRight` allow the turtle to turn 90 degrees clockwise or counter-clockwise. They cannot be blocked, and will not return anything.
- The `place`, `placeUp` and `placeDown` commands allow blocks to be placed in adjacent spaces. If the spaces already have a block, then the placement will fail, and the commands will return a boolean `false`; otherwise, `true` will be returned.

- The `dig`, `digUp` and `digDown` commands allow erasing blocks from adjacent spaces. If the positions are already free, or are outside the training space, the commands will fail. They will return a boolean value, equal to the success of the operation.
- `detect`, `detectUp` and `detectDown` will allow detecting blocks next to the turtle. They will return `true` if a block is present in the given space, and `false` otherwise.

One interesting thing is that most of the commands only allow interactions directly in front of, above and below the turtle (`back` also allows moving backwards). This means that to move into or interact with the positions on the sides of the turtle, it must first turn to face them. Another important consideration is that the operations that place or destroy blocks only affect adjacent spaces, and thus the robot needs to move to a location in order to interact with it.

2.2 The training space

In Minecraft, a turtle can interact with most blocks in the full 3D space, which is effectively unbounded on the horizontal axes. For my purposes, I will use a limited 3D space, only 16 blocks across in every direction. The robot can move only inside the training space, and only affect voxels contained within it. The area out-of-bounds can be considered, for the purposes of turtle logic, to be made up of unbreakable blocks.

Every position in the space can be identified by a triplet of coordinates (x, y, z) where $x, y, z = \overline{0..15}$ and where the y-axis is vertical and the x- and z-axes are horizontal. This is analogous to the inspiration material, as Minecraft also uses *y-up* space. The positions can either be free, or taken up by a block (or voxel). The initial position of the turtle in the training space is $(0, 0, 0)$, and its direction is towards positive x. Every position is initially free, and may be filled by the turtle during its operation.