LOST ON ISLAND

A cooperative automated game

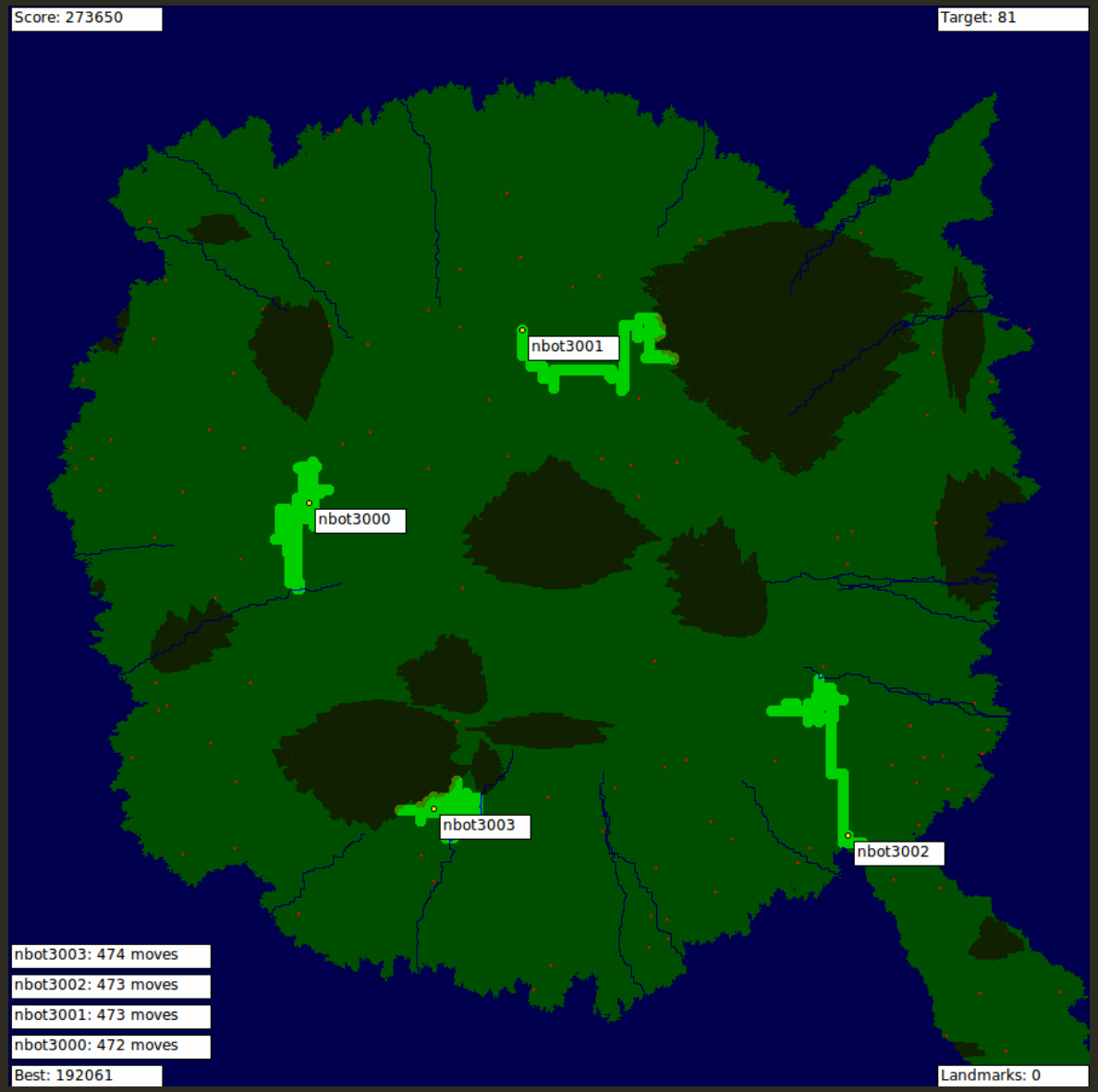
Bruno Guedes

IN A REMOTE ISLAND…

A plane expedition goes awry, and all occupants are forced to jump on parachutes on a remote island.

Now they need to group together in order to be able to be rescued.

Their only communication means is a radio apparel that does not work everytime.


Score: 273650
Target: 81
nbot3001
nbot3000
nbot3003
nbot3002
nbot3003: 474 moves
nbot3002: 473 moves
nbot3001: 473 moves
nbot3000: 472 moves
Best: 192061
Landmarks: 0

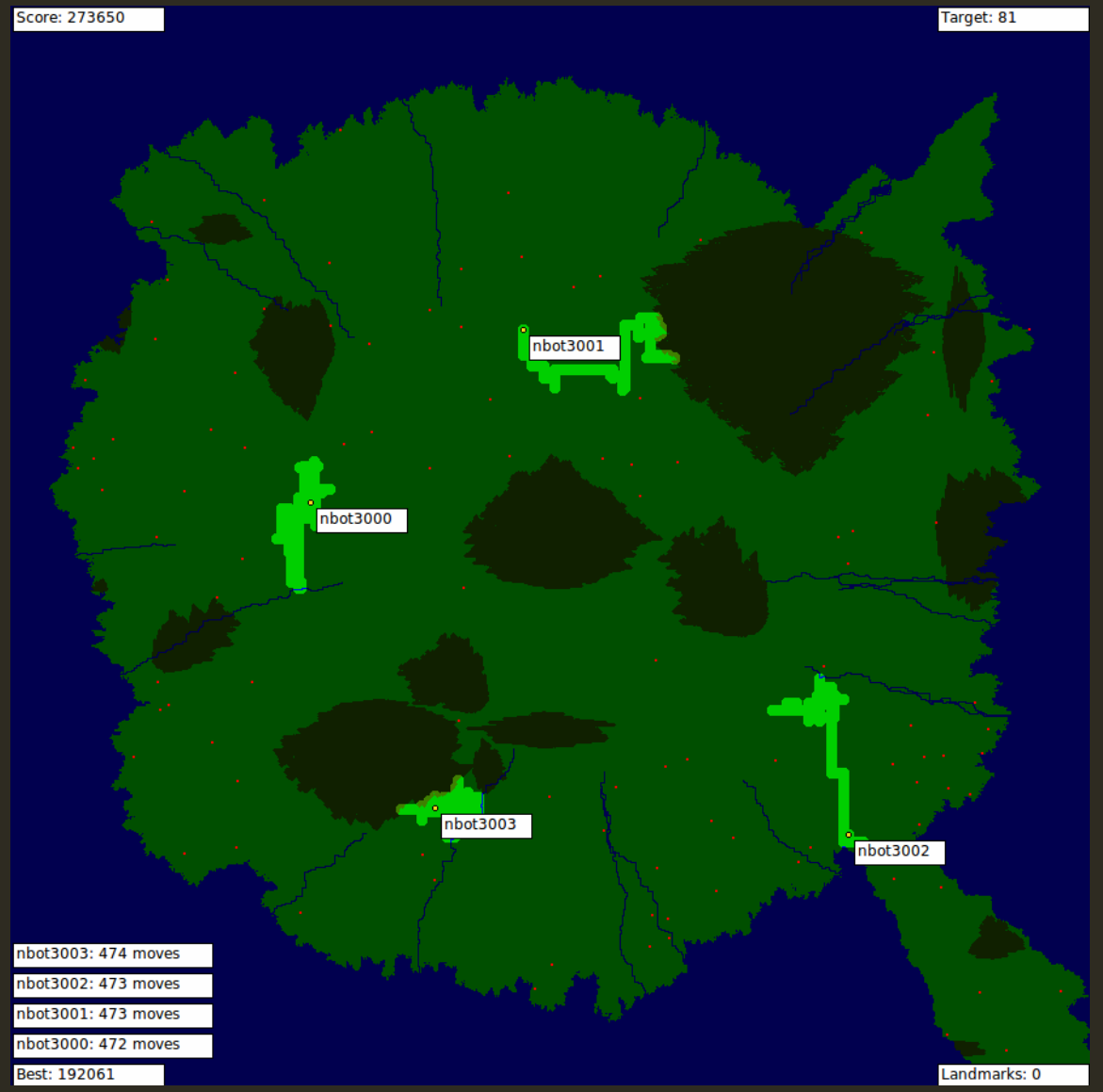
GAME OBJECTIVE

The game sets multiple bots into an island.

These bots have limited vision, they can only see a few steps around them.

They get information about the terrain in the surroundings (water, trees, land) and they can notice unique landmarks in the island.

The goal is to gather all bots into a same location, based on this really limited information.



FLOW OF PLAY

Each group implements one of the bots!

Each bot has to expose a REST API with some endpoints that tell the game how to move it.

Every few seconds, bots can post a radio message that is broadcast to all other bots. It is the only way that bots can communicate with each other on this island.

To help bots to locate each other, the island has some landmarks that have a unique ID. It is wise to keep track of all landmarks found: if one or more bots have found the same landmarks, they can evaluate their relative positions and then meet with each other!

RAILS API

Each group implements a Ruby on Rails server implementing the API routes in the documentation (https://github.com/bsguedes/bot-finder-core)

Try to use the best practices of a MVC application that we use in our project.

Tips:

- Use somekind of DB connector or Rails cache to keep track of information shared between bots.
- Create a PlayersController to expose the API routes
- Create a PlayersModel to implement movement logic.
- A view is not necessary, but could be a useful tool for debugging.

A dummy bot example (in Node.js) can be found here:
https://github.com/bsguedes/bot-finder-player

API ROUTES

GET /players/name

Expects a JSON (application/json) containing the name of the player. Every time this is called, a new game begins.

It is expected that the API returns this payload:

```
{ 'name': 'bot_name' }
```

Please keep the name down to 8 characters.

API ROUTES

PUT /players/:player_name/move

The game sends this payload in this call:

```
{ 'vision': (VisionObject) }
```

Where VisionObject is an integer 2D array, indexed first by x then by y, containing what the bot currently sees. Each field has these possible values:

- -1: darkness (your sight does not reach the corners of the square)
- 0: land (walkable terrain)
- 1: water (rivers and sea; impassable terrain)
- 2: tree (only on land; impassable)
- 100..199 (a landmark; each landmark is unique and impassable)

The expected return value is one of the four possible directions in this format:

```
{ direction': (‘north’, ‘south’, ‘east’, ‘west’) }
```

API ROUTES

GET /players/:player_name/radio

Expects a JSON (application/json) containing data that will be sent to all players, under a radio key.

It is expected that the API returns this payload:

```
{ 'radio': (RadioObject) }
```

Where RadioObject is a JSON object with whatever data you wish.

API ROUTES

POST /players/:player_name/radio

The API posts to each player through this route the contents of every radio signal with the following payload:

```
{ 'player_1': (RadioObject), 'player_2': (RadioObject), ... }
```

All radio messages are broadcast to all players (including the message you sent before)

Radio messages are not necessarily synced.