RobotZ

[Quantal] Robotz - a real-time strategy game

The scope of this project is to implement a simulation of a virtual world for experimenting with different strategies of resource gathering, structure building and combat.

The main protagonists of this world are the *robotz*, simple beings of little need but great might. They live in *[Quantal]* - a world where everything is discrete, matter, energy, even time and space come in quantums.

The [Quantal] world is simulated inside a server program. The state of the world is represented as a suitable data structure, and all modifications to this world is made only by the server.

One or more client programs can connect to the server through a TCP socket. The clients can query the current state of the world and send commands to the server. The server processes these commands and updates the state of the world accordingly. The interaction between the server and client(s) is completely defined by the set of messages they can exchange. The only requirement for a client is to respect this message protocol. However, for this project we need a client that provides a suitable user interface for displaying the state of the world and let the user interact with it. The client must let the user to execute basic operations on the world, like moving an object to a destination, picking and dropping things, start building a structure, etc. The fun starts with the client-side implementation of smart algorithms for resource finding, building placement, enemy scouting and combat. The user interface must let the user to select from and start these predefined algorithms at any time. Just watch the robotz doing their jobs on their own!

An advanced feature of the project is the development of a specialized language in which the user can program the [Quantal] world. The client must implement an interpreter for this language and let the user edit the programs in run-time. The programs can be stored/loaded from disk for later sessions.

For programming [Quantal], a lisp-like language is proposed because of the flexibility and syntactic simplicity. However, the exact specification of the language is open to discussion.

Last but not least, a map editor is required for making different maps on which the battles go on.

The [Quantal] world

[Quantal] is a 2D world with very simple physics. In this way we can focus on elaborating algorithms for plan generation, flexible response to tactical situations, cooperation and coordination between actors, avoiding the complications of real-world physics simulation like forces, acceleration, friction etc.

The virtual world has four fundamental "physical" properties: Space, Time, Matter and Energy.

Space is measured in **dot** -s. [Quantal] is world of rectangular shape having a number of dots in X and Y direction. The positions of objects is given as an (x,y) point measured in dots.

Time is measured in **tick** -s. The state of the world is updated in the server from tick to tick. Nothing can happen between two ticks.

Matter is measured in **gran** -s. [Quantal] has different kinds of resources, like stone, iron, etc. The quantities of these resources are integer number of gran-s. Also the objects have a maximum capacity to store a given number of grans. For building objects there is a need of resources expressed also in grans.

Energy is measured in **erg** -s. Different actions like movement, building, weapon firing require energy. There are also energy sources, like the blue giant star of [Quantal] which provides robotz with solar energy by using solar panels. Another source of energy is nuclear cells built from uranium resources found on the surface or in the soil of [Quantal].

Other quantities can be defined based on the fundamental ones: speed is measured in dots/ticks, power is expressed as erg/ticks etc.

The landscape of [Quantal] is not complicated : there are areas of **ground** and **water**. Some robotz can move only on the ground, others on the water. There is also a possibility to fly: flying robotz can travel in the **air** above any type of ground.

Objects in [Quantal] are of four main types: **resources**, **buildings**, **robotz** and **gadgets**. Only robotz can move. Buildings, once built, keep their position. Resources can be picked or harvested and carried around by robotz. Gadgets complete the picture: they are all kind of utility objects like solar panels, fuel cells, radars and, of course, weapons.

Resources come in two flawors: surface and underground. Surface resources are tiny bits of matter scattered all around on [Quantal] ground. Robotz can pick them and use them for various tasks. Underground resources must be harvested by building an extractor above the mineral patch. The geology of [Quantal] provides the robotz with **stone**, **iron**, **uranium** and **silicon**. Different objects can be built using a specified amount of these resorces.

Robotz have a common structure: they are built on **platform**-s. A platform provides a way to move around, on soil, water or in the air, depending on the class of the platform. Each platform has a manipulator which can be used to pick and drop objects like resources and gadget. The platforms have a **box** where object can be stored. The box of each type of platform can hold objects up to a specified amount of grans. Platforms also have a specified number of **sockets.** Gadgets picked by the manipulator can be mounted on a sockets and in this way became usable. For example, a solar panels or different types of weapons can be mounted on sockets.

Al l robotz differ only by the type of their platforms and the number and type of gadgets they have mounted on their sockets.

Buildings can be created by placing a special type of gadget, called a **nano-fab** on the ground. The nano-fab can be instructed to “grow”, and slowly transforms into a full **fab** or **pylon**. When the nano- fab starts growing, it occupies a certain area on the ground, called **foot-print**. In order to grow, resources must be placed on the fab’s footprint area in sufficient quantities. These resources are consumed in the building process.

Fabs are capable of fabricating platforms and gadgets or extracting minerals from underground. Pylons do not produce objects, but have sockets on which gadgets can be mounted. A good defense can be built by placing pylons and mounting weapons on them.

Implementation

Server requirements

The server must provide the following functionalities:

- Access a list of previously designed maps stored on disk
- Open a TCP server socket and listen for incoming connections
- Manage the list of connected clients
- Manage a list of games, allowing clients to create new games and join to existing games

- Have a chat possibility between clients
- Let clients which joined the same game to start the game
- Handle the commands sent by clients and update the state of the world of a game

In order to do this important data structures must be designed and implemented:

World representation: a suitable class hierarchy must be designed for representing every aspect of the [Quantal] world: ground types, resources, robotz, gadgets and buildings. Methods for manipulating the world must be implemented, such as creating and destroying different objects, moving and building.

Communication protocol : a class hierarchy representing the different messages exchanged by the server and clients must be designed. These messages will be sent and received through a TCP socket.

External Map representation: a way of storing the maps on disk must be developed with methods for loading and storing.

Client requirements

These are the things the client must know:

- Connect to the server by a TCP connection
- Query the server for the list of connected clients and defined games and present this info to the user
- Create or join games
- Start a game or be ready for a game to start
- Handle the communication with the server during a game by sending commands to update the world and receive notifications from the server when something is changed in the world
- Display the current state of the game in a (simple) graphical way
- Provide the user with a means to interact with the objects in a game
- Implement different algorithms for automating tasks in the game
- Let the user activate these predefined algorithms
- Implement an interpreter for a language in which the algorithms are specified
- Let the user edit and test interactively different algorithms written in this language

Map editor requirements

The map editor is a standalone application which provides a graphical user interface for creating new maps. It must use the same representation of the world as is used by the server. The format of disk storage for the maps must be identical to server format.

Last word

The last word on this project is not spoken yet. As you can see, the things told so far are guidelines for designing a world suitable for experimenting with different algorithms. Many details must be worked out, both in terms of specifications as in terms of design and implementation. There is lot of room for imagination and new ideas can be adopted as the project goes on.

We are happy to provide all the help we can give.

Have fun!