Andrew Perrault *10 October 2014*

On The Rocks GDD

Story

The main story is about an astronaut mission to explore an asteroid. As the astronaut attempts to land on the asteroid his vehicle fails and he crashes. The game begins with the astronaut waking up in a heavily damaged ship at the bottom of a deep crevasse. The object of the game is to survive long enough and make contact with the rest of the mission and get rescued. The main conflict of the game is very much man vs nature and as such very little will be explained to the player about the main character. He/She does not even have a name and will have no communication with any NPCs at all until the final sequence when you actually make contact and win the game.

Puzzles

At the start of the game everything is dark except for a red blinking alarm signal.

There is a flash light and a communication station on the ship. Leaving the ship without the spacesuit will result in you dying. The communication station is working but cannot get a signal from the bottom of the crevasse.

The ship main power is broken and you cannot turn on the computer. Take the fibre optic cable from the cargo hold. Open the locker and put on the space suit before leaving the ship.

Take the antenna from outside the ship. Behind the ship you will find an access hatch. Open the hatch and take the solar power generator and the power conduit.

Enter the tunnels.

You find a huge crystal grotto. Light from the sun is refracted through the crystals and this room is the bright. Drop the solar generator. Attach the conduit to the generator.

Go back to the ship. Attach the wire to the auxiliary power generator. The computer will now work.

Exit the ship. Take the antennae.

Go back into the tunnels and find your way to the surface.

Find the top of the ravine. Setup the antennae. Attach the fibre optic cable to the antennae. Throw the cable into the ravine. If you try to plug in the cable and walk through the tunnel the cable will get unplugged.

Go back to the slope behind your ship. Take the cable end. Go to data port on the side of the ship. Attach the cable to the data port.

Use the computer. Contact the rest of the mission crew in orbit around the asteroid.

Win.

Interface

The player interacts by typing commands at a prompt.

n, s, e, w, up, down can be used a short forms for go north, go south, etc

inventory or i displays the players inventory

look or l displays a description of the current location

examine or x displays details of an object

get and drop picks up and drops items

help will display instructions

quit will exit the game

Technical Systems

Game Object and World Class

All items, rooms and the player will derive from the game object class. The World class is the base of the object tree and also contains add, move and get functions for the various game objects. The game objects will be related to each other in a tree like structure. Each object will have one parent that may be null – rooms will have null parents. All objects will have a vector of children which indicates the game objects contained within that object.

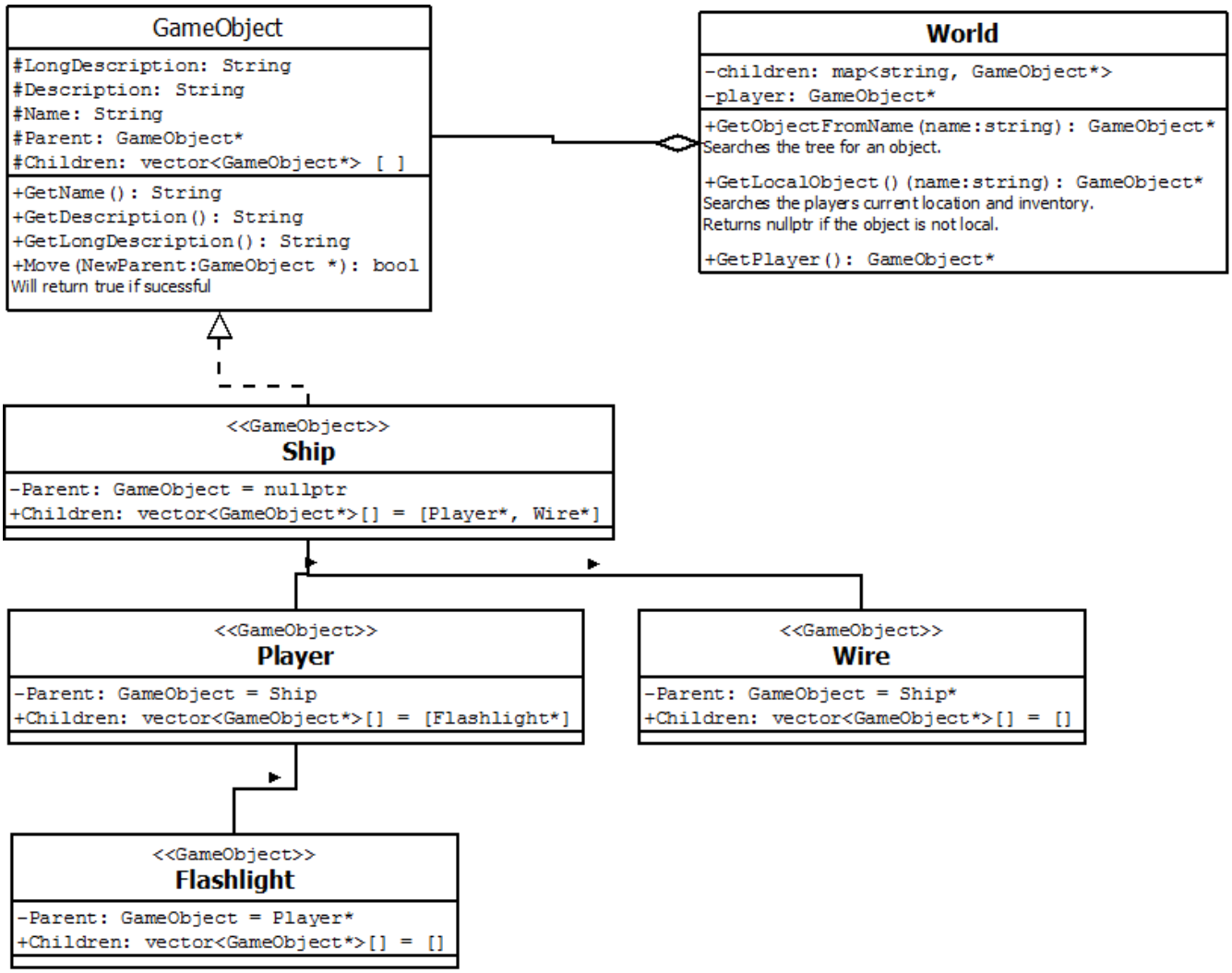
Few examples

1) The ship interior object has two children - the player and a roll of wire. When the player picks up the wire the wire node is moved from a child of the ship to a child of the player.

2) When the player goes to another room the player's node will move from a child of one room to the child of another.

Many if not most of the actions within the game can be completed by reading or manipulating this tree structure. If you want to use an item it must be a child of the player node or a child of the players parent. Changing rooms is just moving the player node. Picking up items is moving the item node. Generating a description of a room would be the players parent description with its child descriptions appended.

World and GameObject class structure:



Tokenizer

All input to the game will come from the player typing commands at a prompt.

Some examples of commands would be:

Go north

Inventory

Open Door

Take Wire

The tokenizer is built into the grammar tree class. It is the class that reads in the commands in English and converts it to a form that can be used by the game. The lexer is not intended to be a natural language processor. The puzzle in a text game should not be trying to figure out which words the game understands and the lexer will value getting the most likely intentions of the player over grammatical correctness.

The number of verbs that the game understands will be vary limited but there will be a much larger

number of aliases to these verbs. The verbs the game understands will include Go, Put, Get and Use.

The tokenizer will read in the words of the sentence and replace the words with a token. Token will come in a number of types including verb, noun, list separator.

For example:

The player types: ***Take wire and antennae***

The lexer will search a dictionary and find take is an alias for the 'Get' verb. Wire and antennae are both nouns also found in the dictionary.

The command would be translated to : **{verb:Get}{noun:wire}{listSeparator}{noun:antennae}**

The verb will then be used as the key in a map of verbs to function pointers and the function Get will be called with an array of items as a parameter :**Get([{noun:wire}, {noun:antennae}])**

Many words like 'with' and 'on' will be ignored.

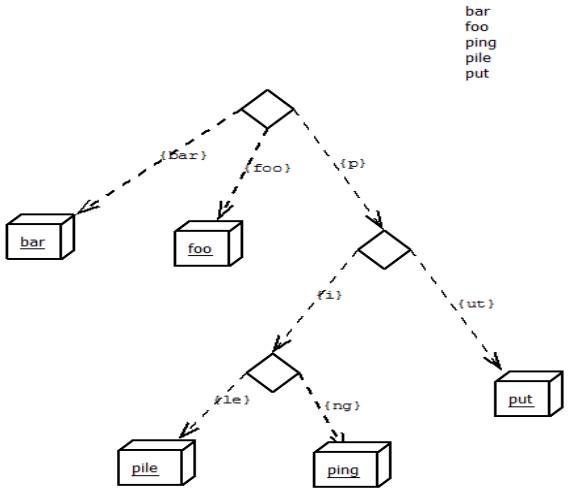
Use key on door and ***Open door with key*** and ***Use key door*** would have identical translations of:

Use([{noun:key},{noun:door}])

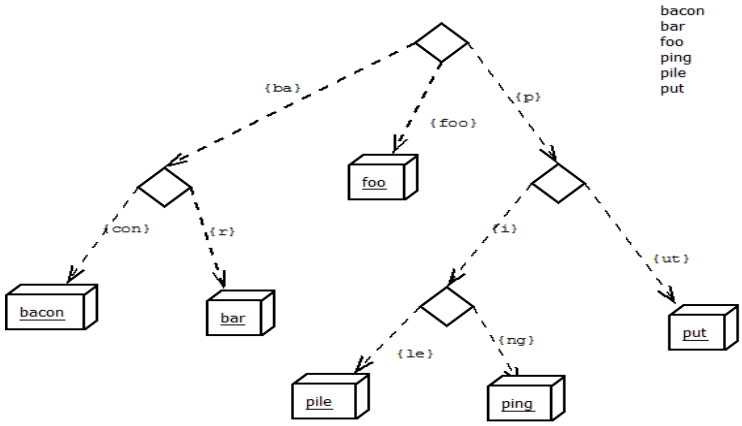
Obviously ***Use key door*** is not grammatically correct but ignoring small words makes the code much simpler and that is most likely what the player intended anyway.

I will need to keep an eye out for specific edge cases where the system may break down. For example the lexer should not confuse ***Pick up rock*** with ***Go up*** or even just ***Up***.

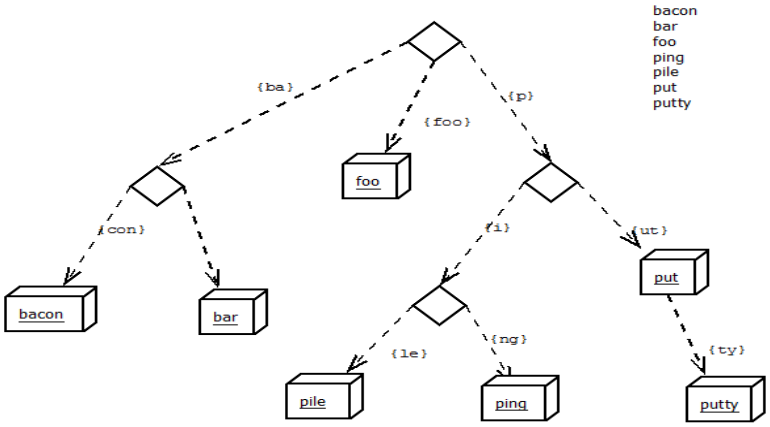
Grammar Tree Structure:



Add bacon

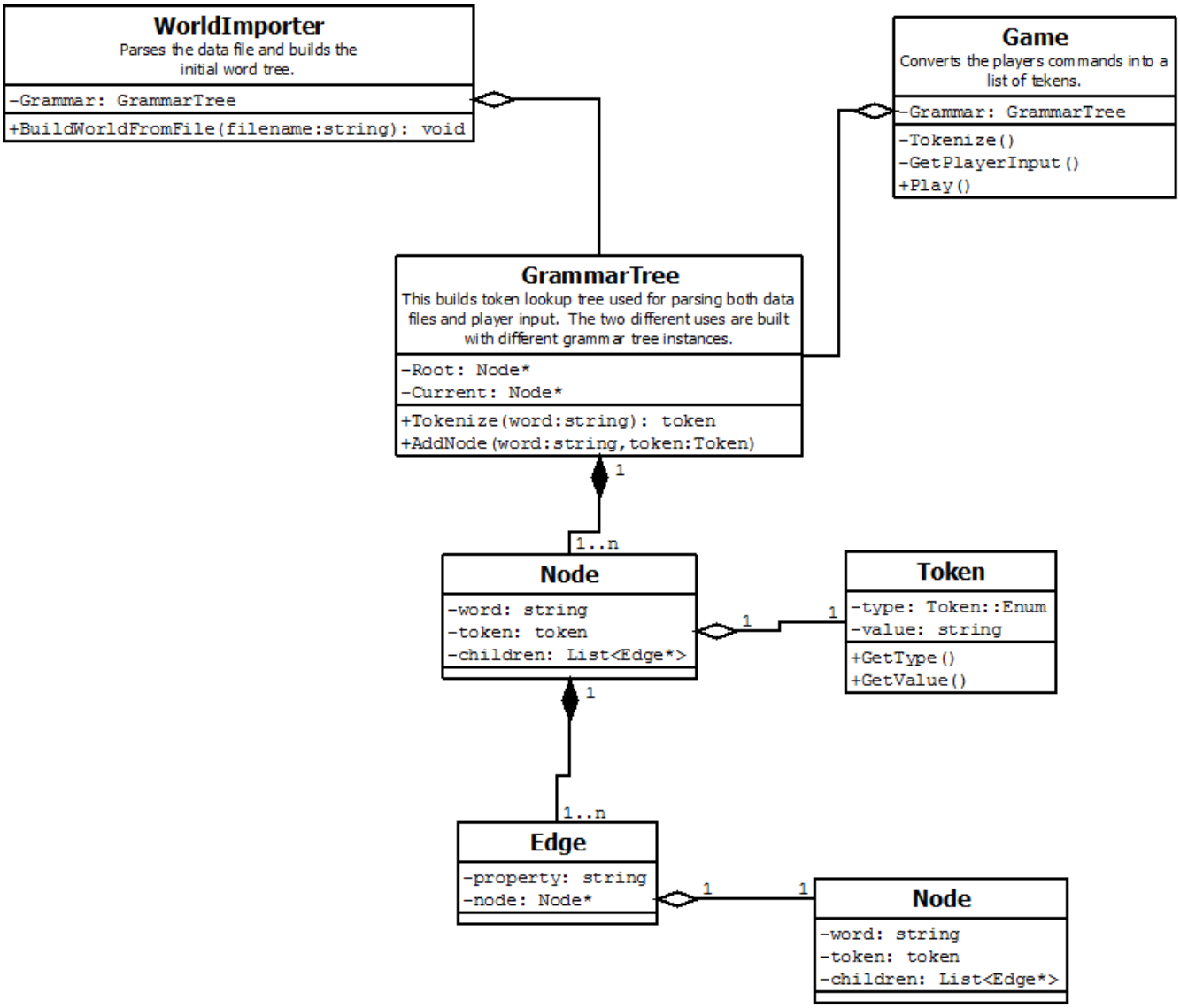


Add putty



{r}

Grammar Tree Class Diagram:



WorldBuilder

The game objects will be imported from text files rather than hard coded. The files will have the object name and a list of properties and values on each line. The room importer will make use of the tokenizer with an alternate grammar dictionary to interpret the text files.

1) First the tokenizer will make convert the file into a list of tokens.
2) Next a syntax tree representing all of the game objects and their properties.
3) The world builder will then instantiate the game objects.
4) The last step is to do a second pass through the syntax tree. On this second pass all of the game objects relationships will be updated. Room exits will be built and the structure of which objects belong in which room will be built.
5) One last function of the world build is to add all of the game objects names and aliases to the

game grammar tree as nouns.

The intention of the room importer is to make adding content to the game much quicker than having to rewrite headers or building long list of hard to maintain code for every new object. The object importer will build the game object tree and also add a token to the command interpreter grammar tree automatically on import.

Data File Example:

```
name : "player" {
    type : player,
    description : "player",
    long_description : "This is you"
}

name : "cockpit" {
    type : room,
    description : "Cockpit",
    long_description : "You are in the cockpit of a spaceship. There is a strong smell of burnt electronics. Everything is dark except the dull glow of red emergency lights. There is a doorway leading south.",
    exits : {
        name : "small_corridor" {
            alias : "door",
            alias : "south"
        }
    },
    children : {
        name : "computer" {
            alias : "computer"
        },
        name : "socket" {
            alias : "socket"
        },
        name : "player" {
            alias : "player"
        }
    }
```

}

name : "computer" {
    type : object,
    description : "a computer",
    long_description : "On the wall is a small computer."
    detail : "The computer seems to be a communication station."
}

name : "socket" {
    type : object,
    description : "a power socket",
    long_description : "The computer has a socket labeled auxilliary power on the front of the panel."
    detail : "The socket has nothing plugged in."
}

Syntax Tree Example: