Dylan Kobayashi
ICS 661
Final Report

Quasi GPS

Introduction

While mobile devices increase in popularity, so too are the appeals of apps that are aware of the user's location. Having information tailored to a user makes the user experience more friendly. But for some apps knowing a user's location is necessary to work. One such example are services like MapQuest and Google Maps, where a user can input a desired destination and get not only the location, but how to get there as well. The goal of my project was to create a quasi GPS that is able to parse a user's input and correctly move the user's icon to the specified location. In this format it doesn't quite make a good GPS, since most people want to know how to get to the location, not setting their location to the specified destination. In terms of a practical usage GPS application, instead of moving the user's icon, it would give a route instead. My reasoning behind moving the icon is that I knew I would not have enough time to be able to implement a real Geo-coordinate system. Moving the icon was done with the assumption that users want to get to their destination and probably wish to ask for more directions. Without moving the icon, I had no way to change their current position from which they ask for new directions. In theory, if the icon was able to correctly get to the specified location, then it should be able to trace the route.

Depending on the focus, the Quazi GPS applies to a couple different areas. The first being something similar to MapQuest and Google Maps, where a user can ask for directions and it will give a route. As it is currently designed, with a relatively small amount of alteration it can be made to trace routes instead of changing the current position of the user. Taking this a step further is that voice commands can then be implemented. If the parse is completed, it can then work with voice recognition software to take user voice commands, rather than having to type them out. This would be preferable as most people who want directions are usually driving.

Diverging a little, there was one other possibility that could be done with the system. Working with directions and location could be a stepping stone to modeling an AI. This could be one part of a larger AI system. The different usage applications could then apply, to autonomous robots, modeling a population, or even game AI.

Constructing the base system

Thankfully there were some preexisting libraries to handle some of the base requirements, but I had to write something to handle nearly every seen aspect. Everything from the display to the actual mechanics I had to write some amount. The displaying of the elements was perhaps the easiest but very time consuming. The library I used worked with OpenGL and came with a couple downloadable sprite manipulator classes. However, I had to figure out how to position and move them. Positioning was strange in the sense that the monitor's origin point is the top left as opposed to the bottom left. Drawing was also done with respect to the top left corner of an image. Movement in real time was then

dependent upon the objects having a speed which determined how far it moved with respect to time. The difficult part I quickly discovered was that the cycle time was never consistent, so I needed a way to calculate exactly how much time passed to prevent the icon from moving at inconsistent speeds.

Java has the build in swing libraries, but there was difficulty with using them because OpenGL controlled the entire active window. Swing would require a different window and that caused a bunch of update problems. I ended up creating my own input detection class. It only checked for alphas, space, backspace and enter. I did not check for more because implementing those alone was very time consuming. My input detection class operated off of three states: down, just pressed, and just released. In order to work correctly, I needed to also make sure I updated(polled for information). Because a different class stored the user input, the polling and button checking had to be done in a proper order otherwise the status of the button was either “down” or “not down.” The “just pressed” and “just released” were essential to gathering input because the system ran in real time. If the button check was only based upon whether or not a button was “down,” unless a person was really quick, the system almost always ended up believing that the user was inputting multiple of that same button.

Constructing the mechanics

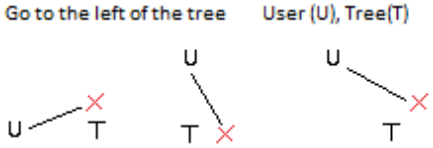
When I finally got the base system setup, I did a simple parse using finite state automata to check for exact structure: [Verb] [to] [the] [Noun]. Where Verb was from a predefined set of verbs(go, touch, walk, run) and Noun also had to be from a predefined list of words that related to the visible objects(tree, bush, plant, shurb, etc). Although this worked, using a FSA was rather cumbersome and didn't scale well. I tried to switching over to context free grammar(CFG) in an attempt to reuse code from assignment 4. I very quickly found that this was a bad idea as any sentence that didn't parse just ended up outputting: “invalid structure: [sentence parse],” the error I defined when the system could not parse something. The issue came as result of the CFG being too strict. If anything was encountered that did not fit nicely, then it would error. Although good on one hand, incredibly frustrating because many sentences I felt should have been parsed, didn't parse only because I didn't have all of the base structure to account for small variations in grammar. Furthermore, any undefined grammar would end up failing the parse because CNF wants to correctly categorize all words. In the early stages of testing, I needed my system to simply work. I also wanted to keep some leeway because people can speak different ways and be grammatically correct. At this point in time I ended up creating a hybrid between FSA and CNF where a user could follow a format but use unknown words or extra words. The parse did a best case scenario that just ignored values it didn't know what to do with.

Part of the problem was that I had not implemented periods(.), commas(,) apostrophes(') or any other punctuation symbols besides spaces. As an example, the sentence “go to the tree” makes perfect sense and the parser could handle that no problem. But when it got to compound values like “Go to the tree, then go to the fountain”. The commas wouldn't be possible to type. When the parser tried working with it, I would detect more than one sentence. But a sentence shouldn't contain multiple sentences. Also, if multiple parsings were obtained, I needed to make sure it was looking for the parse that correctly “looked” like a command.

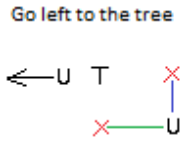
At this point with the parse working at least somewhat decently I started to implement the positional and relational movement mechanics. As stated before, what if I wanted to give command “Go to the tree, then go to the fountain.” Even if the parse could handle it, my system could not. At the time it could only handle one destination at a time. The quickest fix ended up being implementing a checkpoints system. The parse would activate and create checkpoints where the icon needs to move to. After reaching a checkpoint it would continue through the rest.

Going to the tree is fine, but for a more practical system, it would be better to go to the [front/ north/ left / southeast / etc] of the tree. The map relational values were fairly easy to implement, as it

always had a specific modifier based upon object location. For example the north of [noun] always has a y-axis modifier. The difficulty came from user dependent coordinates. For example, if I want the user to go to the left of the tree, that doesn't always mean the west of the tree. This required positional calculation and presented some difficulties since I used rectangular images.



The other positional calculation I used was a direct move reference. For example, when given the command “go left to the tree,” the icon should start moving left of whatever way it is oriented. A couple of support function needed to be created, but overall not too difficult. The real issue was that there was the possibility of ambiguity. What I mean by that is, what if there is no tree? The command “go left” works and the icon will travel left until told to stop or encounters the boundaries of the map. By that reasoning, it was my opinion that the “Go left to the tree” should operate in the same manner. Continue left until encountering a tree, however it will not go past a tree.



The last major feature that I was able to implement was an way of avoiding objects. This presented a large number of difficulties because of the mechanical nature the different parse possibilities. The command “Go the the tree while avoiding the fountain” is fully contained in one sentence. However, an avoid doesn't have to be part of a sentence. For example, I can say: “Go to the tree. Avoid the fountain.” There are also different ways to portray the meaning of, avoid. Other cases can be “dont touch the [noun]” and “stay away from the [noun].” I am sure there are other ways that I missed as well. Getting to this point proved that my current parse was insufficient. The final form of my parse was a mix between CFG and event representation.

No matter how lenient, there is a base vocab that needs to be adhered to in order for the parser to process commands. Exact grammar and syntax are hard to implement without needing to account for many different vocab words and sentence structures, many of which I may not be able to think up. I ended up creating a hybrid parse that first converted words to major part of speech then based upon the word types encountered further checks would be done. Some with more detail other based upon the necessity of word types. For example, proper structure would prohibit a [noun] existing by itself. Each of my nouns needed to have some action performed on it. As an example, “go to the tree then go to the fountain” are valid because the nouns are paired with verbs. But, “go to the tree then the fountain” would not be valid because fountain is not able to pair with an action. Although grammatically correct, my parse isn't able to handle it.

The parser is able to handle “walk to the left of the tree” and shows the icon moving to the tree with a left displacement based upon starting location. It doesn't make the user move left from their beginning orientation. “walk left to the tree” will move the icon in a left direction based off of orientation until the tree. Note the following ways to use an avoid:

“walk left to the tree avoid the fountain”

“dont touch the fountain move to the tree”

“go left stay away from the fountain stop by the tree”

Each of these commands(last I checked), do result in the same action. Move left based off of starting orientation, don't let the user icon move over the fountain, if encounter a tree, stop” They are

able to work because of the event handling which actively tries to pair up nouns with actions. A major thing to note is that in the current implementation it is only possible to avoid one object during a move. This is because my event pairing currently pairs only one avoid with one command.

As described throughout Constructing the Mechanics section, there are a number of hacks that have been performed to get things working. Given more time and resources, the first thing that I would change is from a checkpoint system to an action based system. As is, when a command is given, everything is processed with reference to where the icon is at the time of command. Meaning if I give a multi part command and end with something like “walk to the left of the tree” it will be on the left side with respect to where the icon was when the command was given, not with respect to where the icon would be when it reaches that command. I would also have like to add in valid move areas, or predefine the map with move zones and noun zones, much like a neighborhood grid. The third thing I would like to change is that the input currently doesn't handle punctuation. Finally, I would like to change the parser into a fully implemented system maybe a fully realized CFG or frame based. The hybrid system I have was only done to save time which I was very short on. I am not sure what tools I would use given the option, more research would have to be done before I can give a solid answer.

Analysis:

The system parses correctly the following structures:
[verb] [noun] – moves to the noun
[verb][direction to move] [noun] – moves in direction with respect to orientation. Stops if encounters noun
[verb] [destination position of][noun] – moves to noun with offset based off of position.
each of these structure can then have one [avoid][noun] before, in the middle, or after.
From there each of the structure can be combined to form compound commands.
Unique commands:
“reset” – randomizes the board and clears out all checkpoints
[stop] – will clear out all checkpoints, creating the equivalent of stopping.
[verb][direction] – moves the icon in a direction with respect to orientation.
“barrelroll” – performs a barrel roll.

All part of speech, noted by brackets [], must be defined explicitly in vocabulary.

The current implementation doesn't take into account valid movement paths. So although it will be able to correctly get to a noun and the position desired, the direct path will be direct with the option to avoid one object. Due to this, it could be argued that the current implementation doesn't meet the goal, as it is missing a fairly important aspect that similar systems like MapQuest(MQ) and GoogleMaps(GM) are able to handle.
One thing that is unique to this system is positional relative destinations. “Walk to the left of the tree” and it will go to the left based upon starting location. This usually cannot be done with navigators like MQ and GM because most addresses can only be accessed from one location(street address).

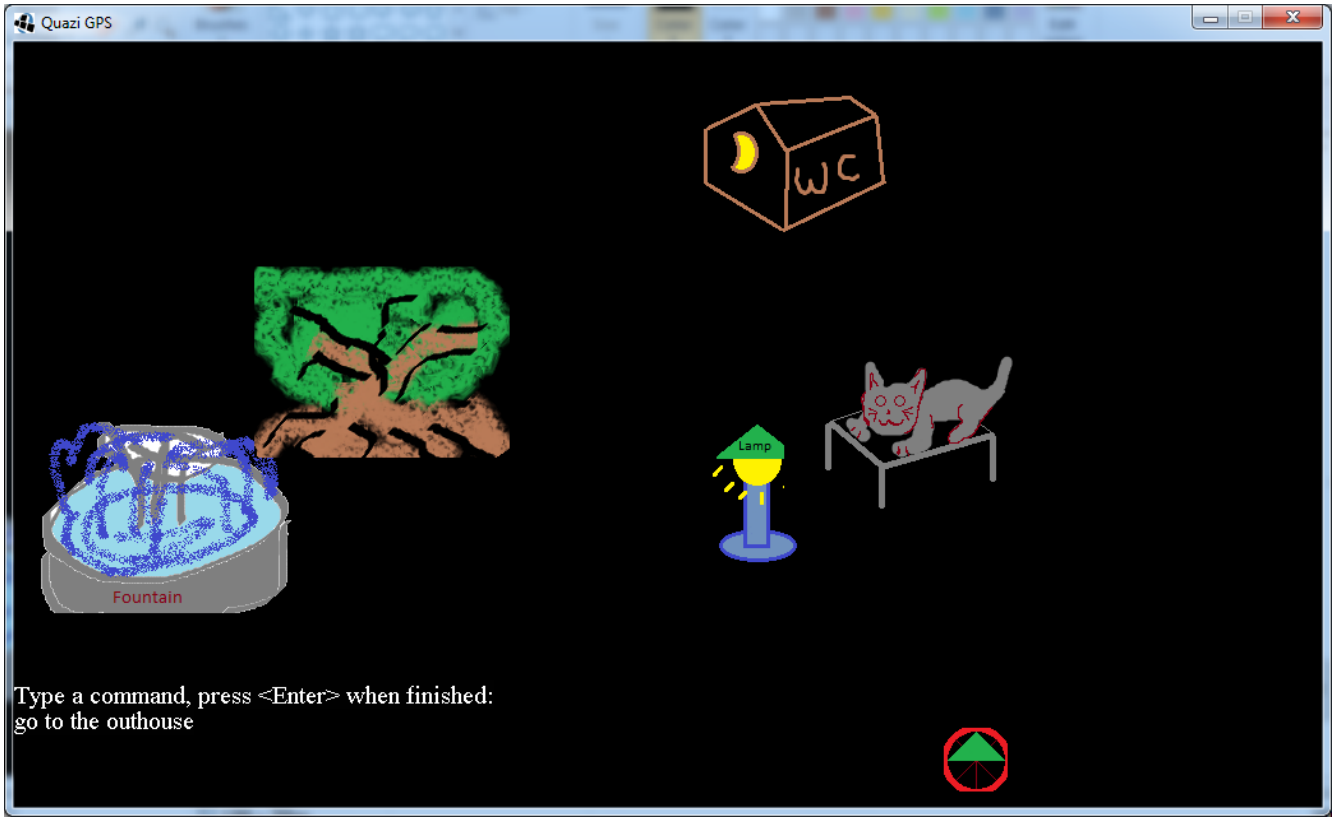
Conclusion

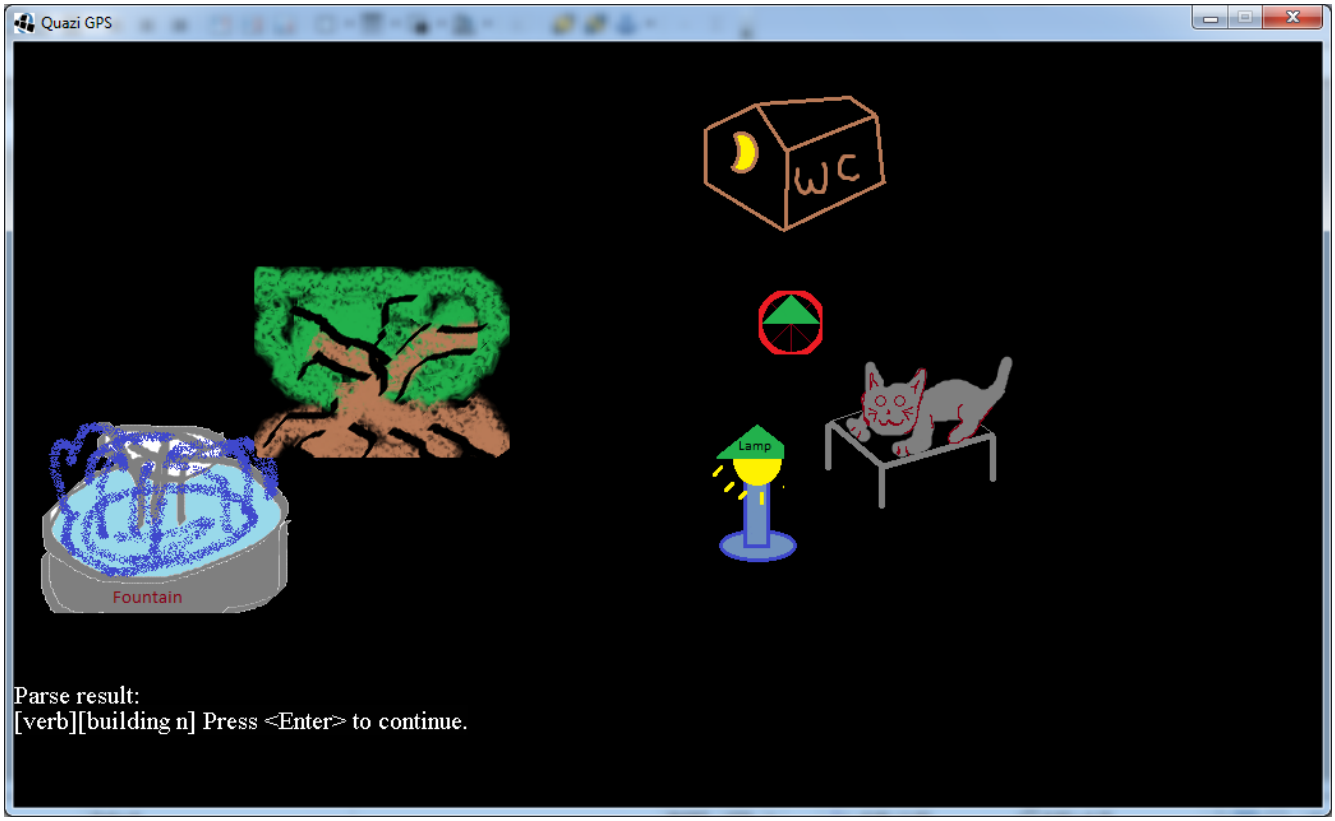
I learned many things from this project. Admittedly, most were not about parsing. Creating the system from scratch may have been a bad idea considering how much time I spent just trying to get

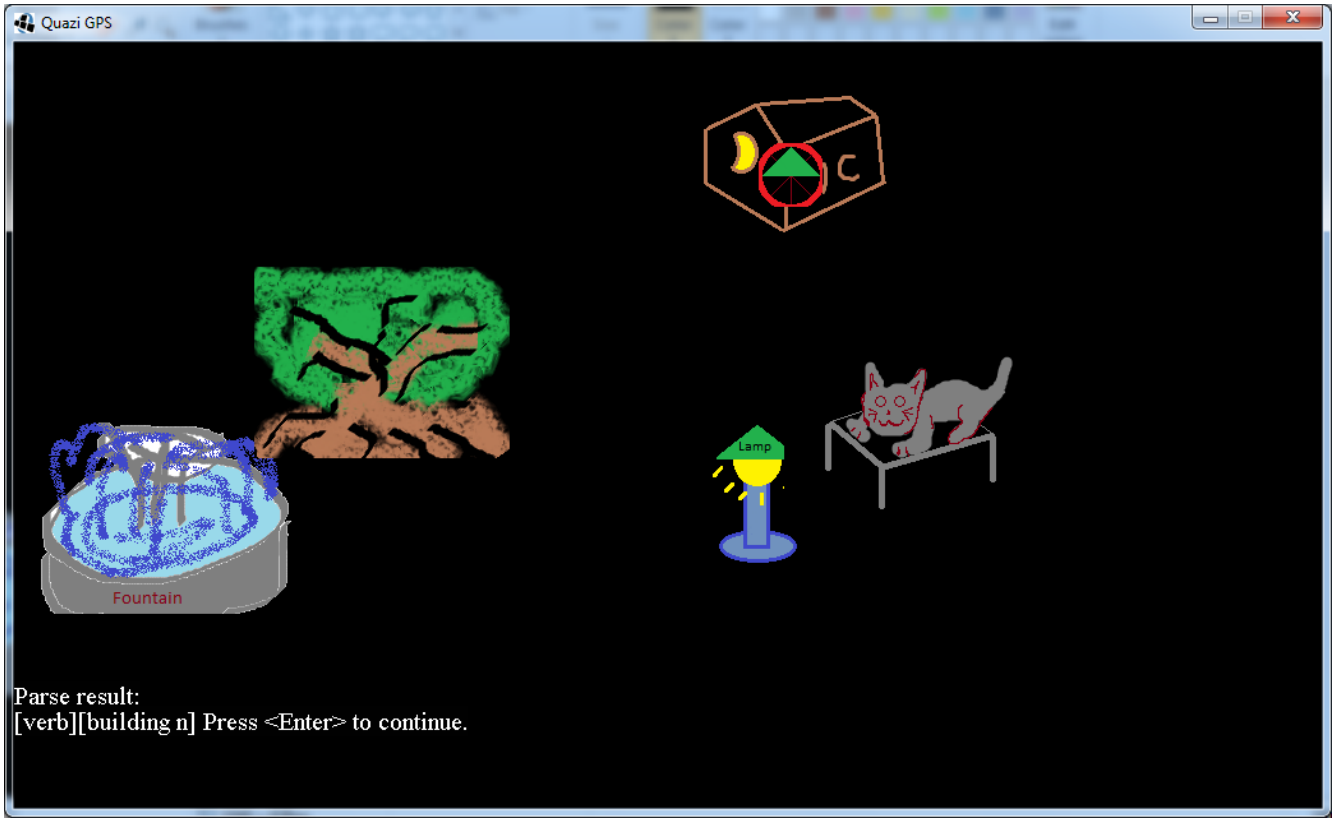
things in a basic working state. Even after getting a basic state established, I need to continually update the system to then handle the grammar and structure possibilities. Through trial and error I learned much about design, getting user input, and the two dimensional coordinate system. When I started to implement the parsing component, I found that highly detailed and specific parses don't work too well when I don't have a fully fleshed out vocabulary. This forced me to create a hybrid system that probably works nowhere near as well as a properly implemented parser. But the problem still remains with what and how to work with the parsed sentence. Since a sentence can be arbitrarily long, whatever converts the parse into actions needs to work in a recursive fashion. Also, a sentence may be parsed in different ways, so the parser needs a validity component, that may be dependent upon explicit knowledge of the map. For example, “go to the green house” may be parsed correctly. But only knowledge of the map would be able to validate whether or not there is a green(adj) house(noun) which the icon can move to. Parsing is very complex in and of itself, applying additional constrains only increases the difficulty.

Appendix:
Sample Run:

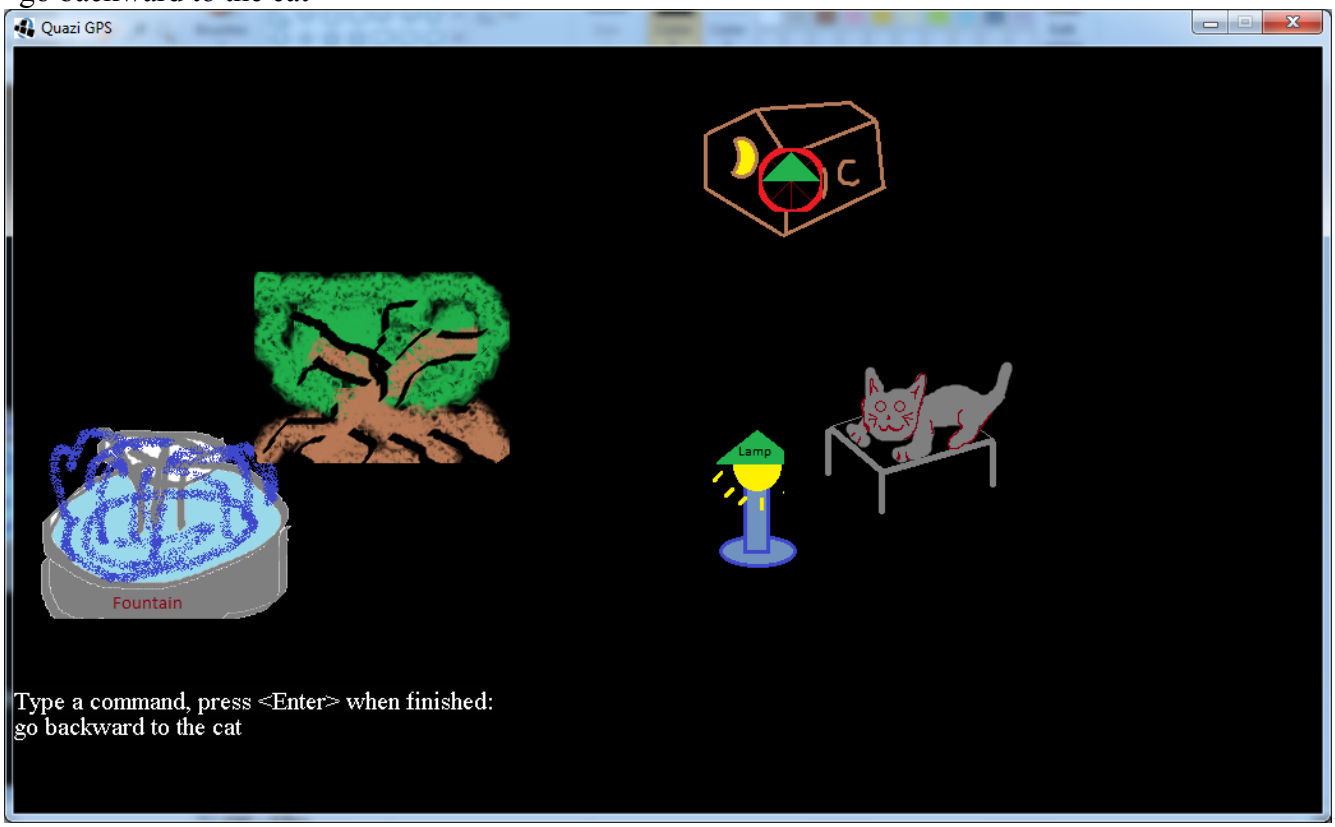
“go to the outhouse”

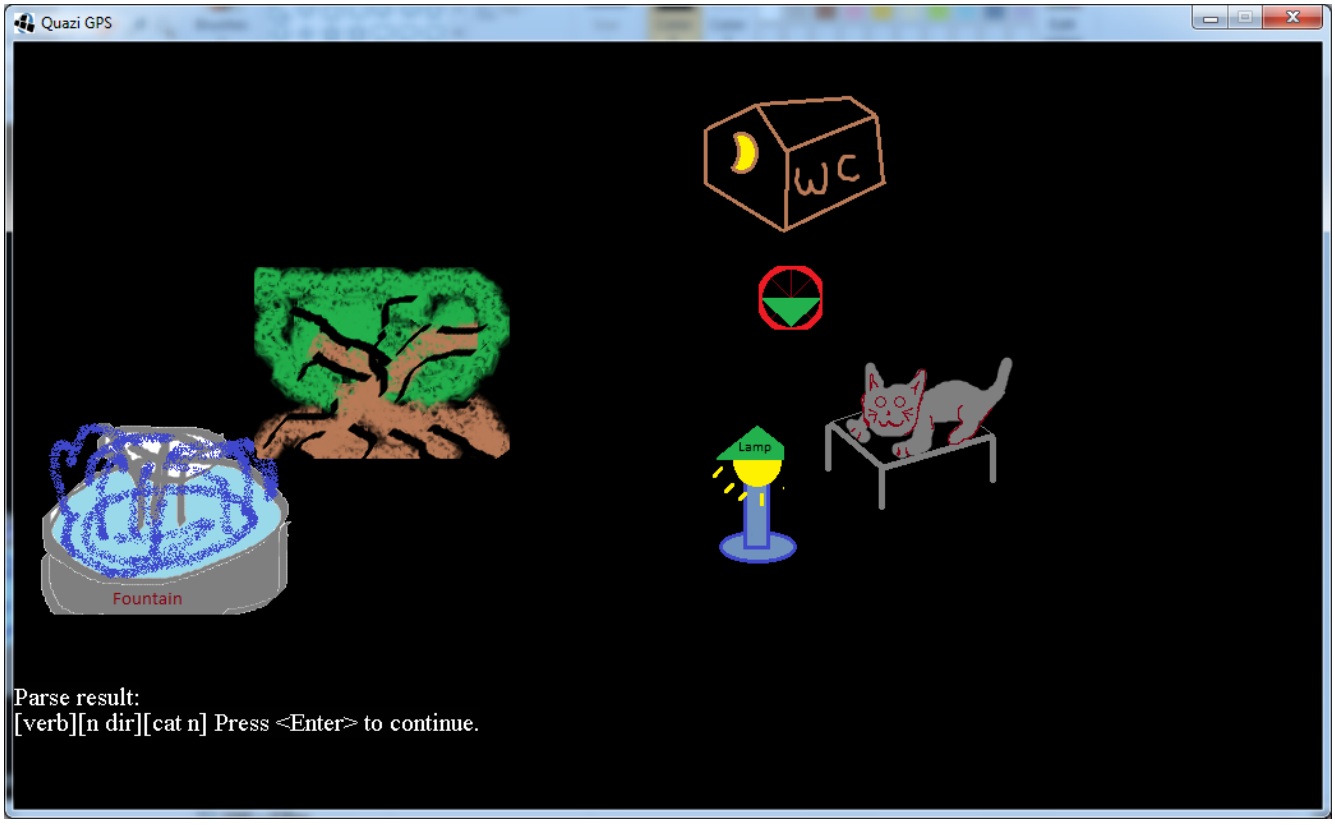

Quazi GPS
WC
Lamp
Fountain
Type a command, press <Enter> when finished:
go to the outhouse

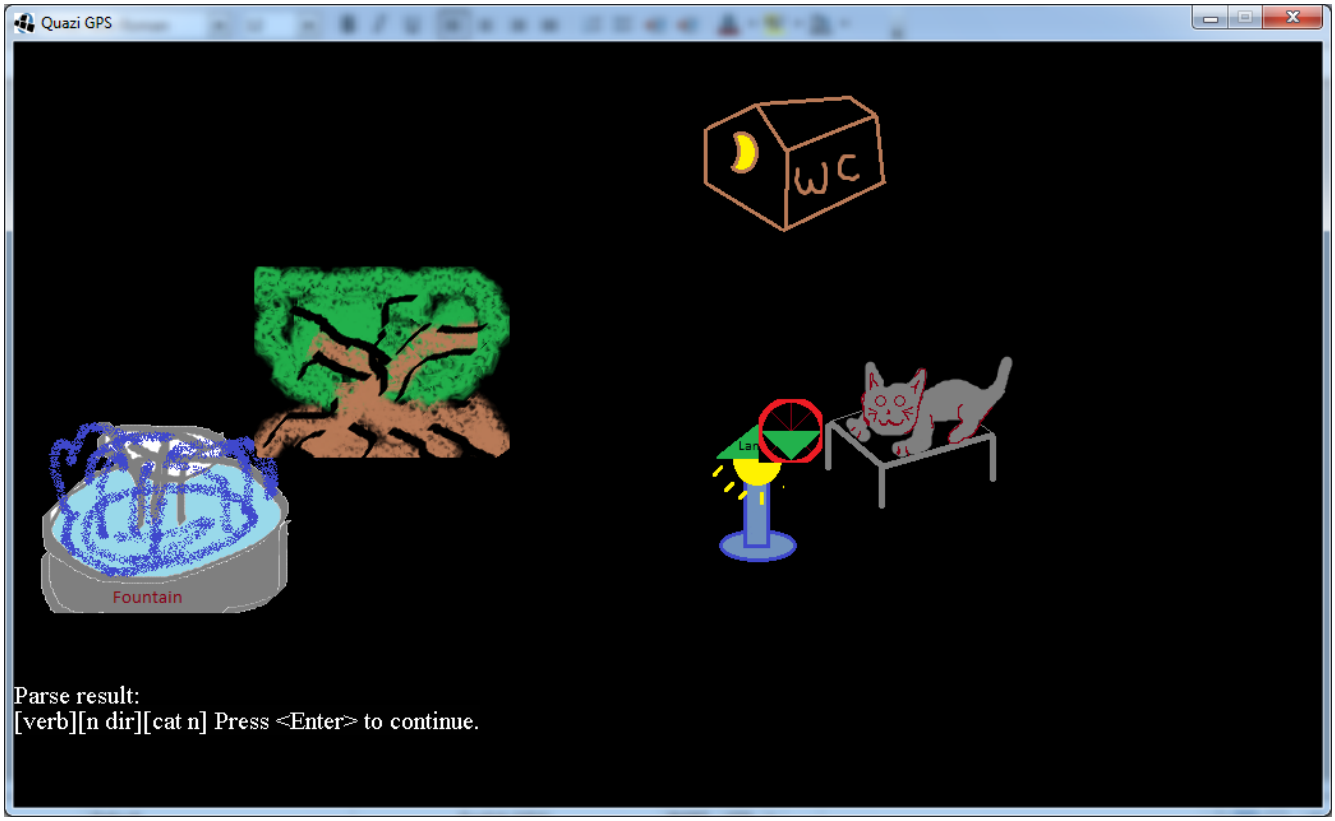

Quazi GPS
WC
Lamp
Fountain
Parse result:
[verb][building n] Press <Enter> to continue.


Quazi GPS
C
Lamp
Fountain
Parse result:
[verb][building n] Press <Enter> to continue.

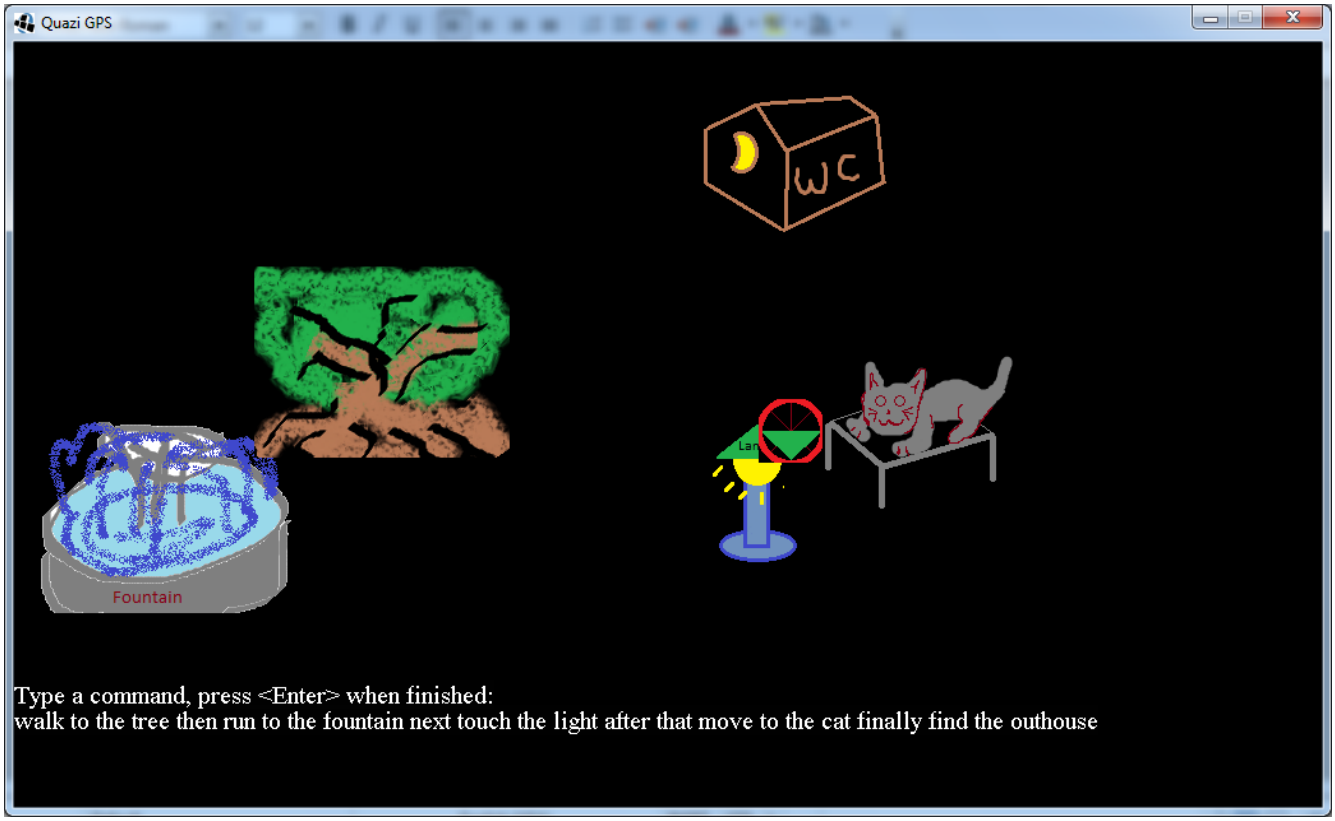
“go backward to the cat”

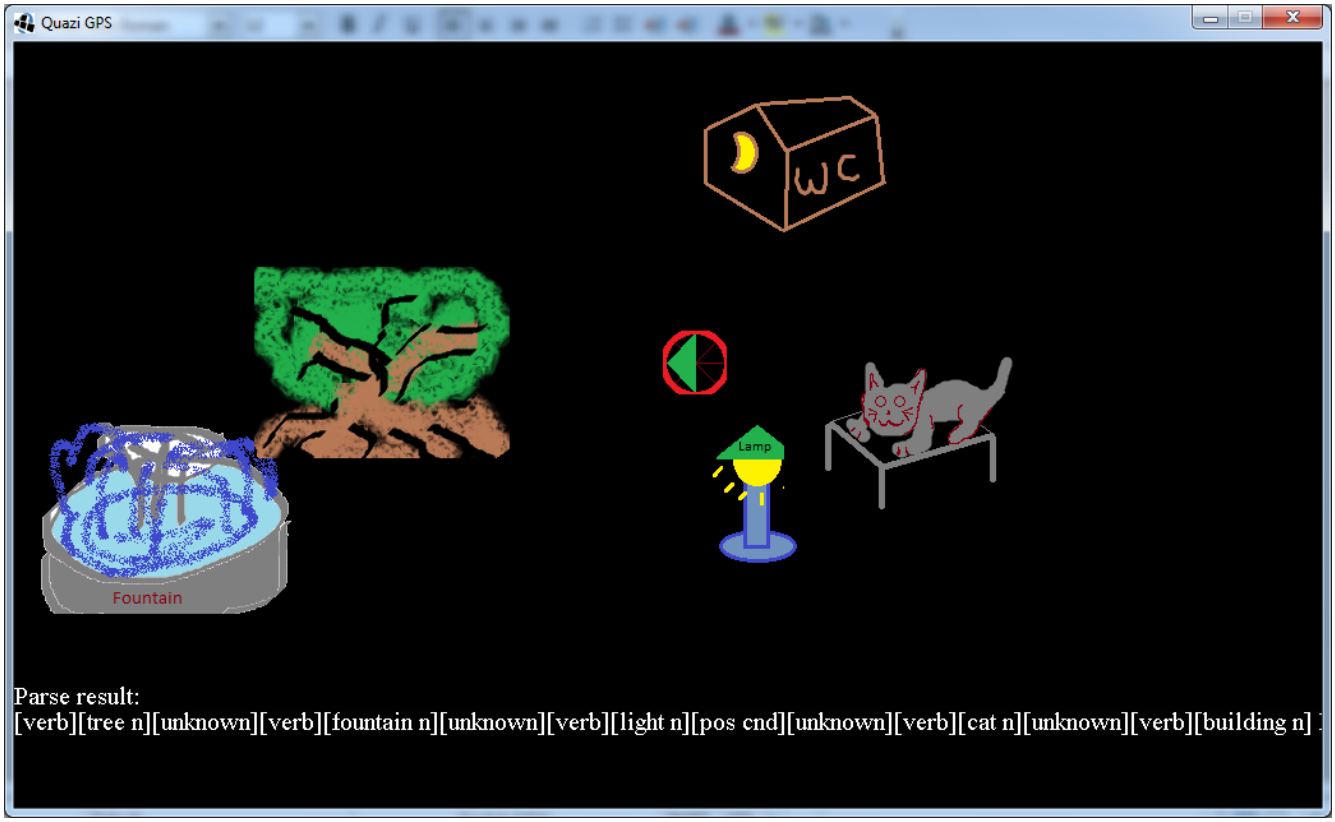


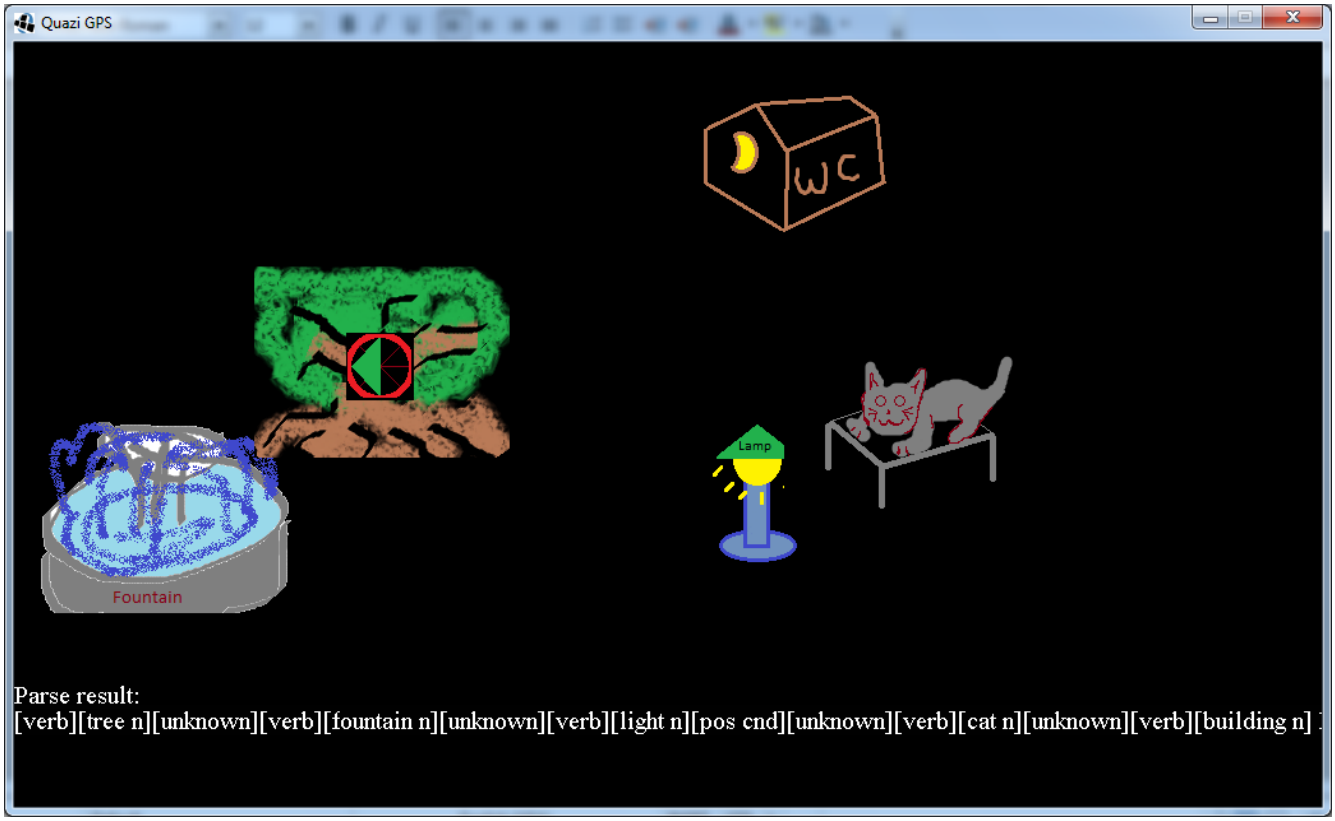


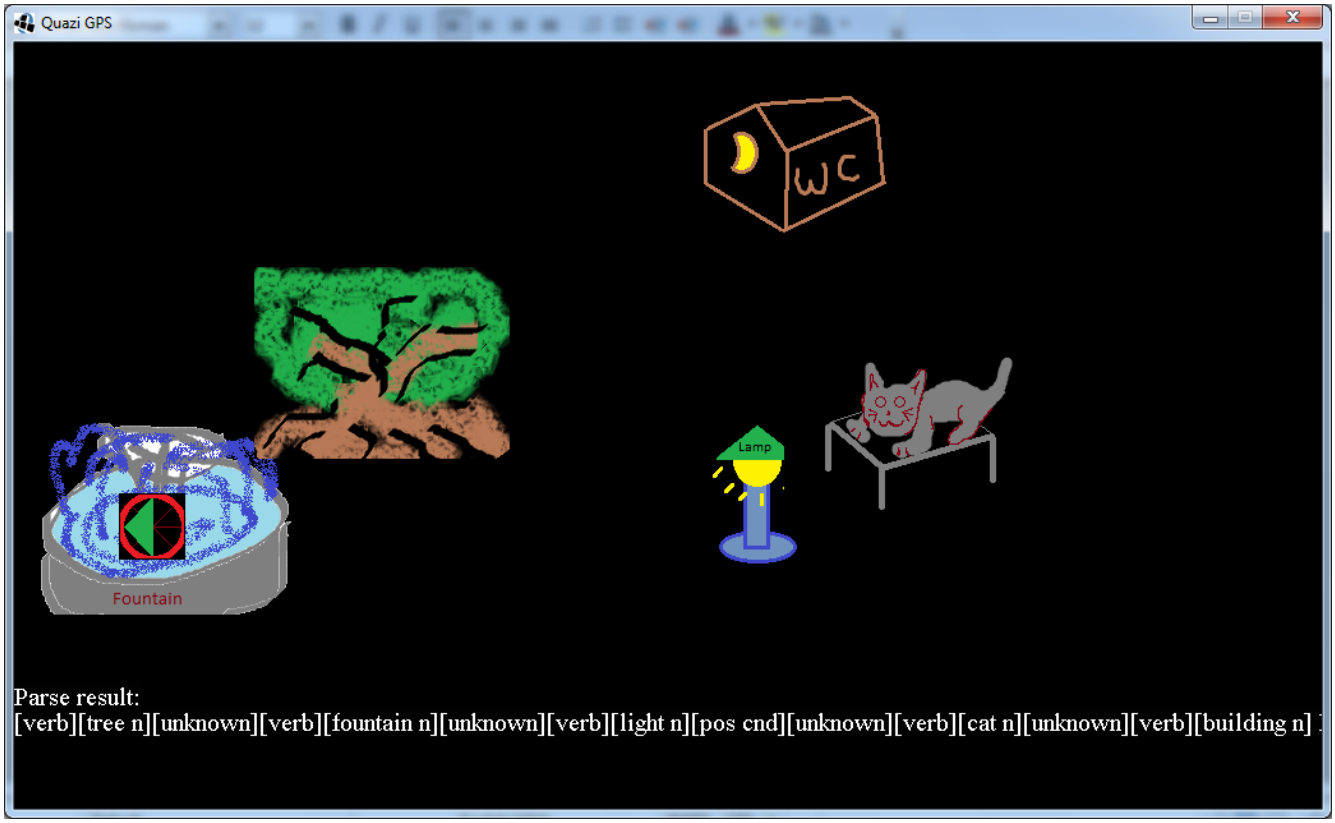


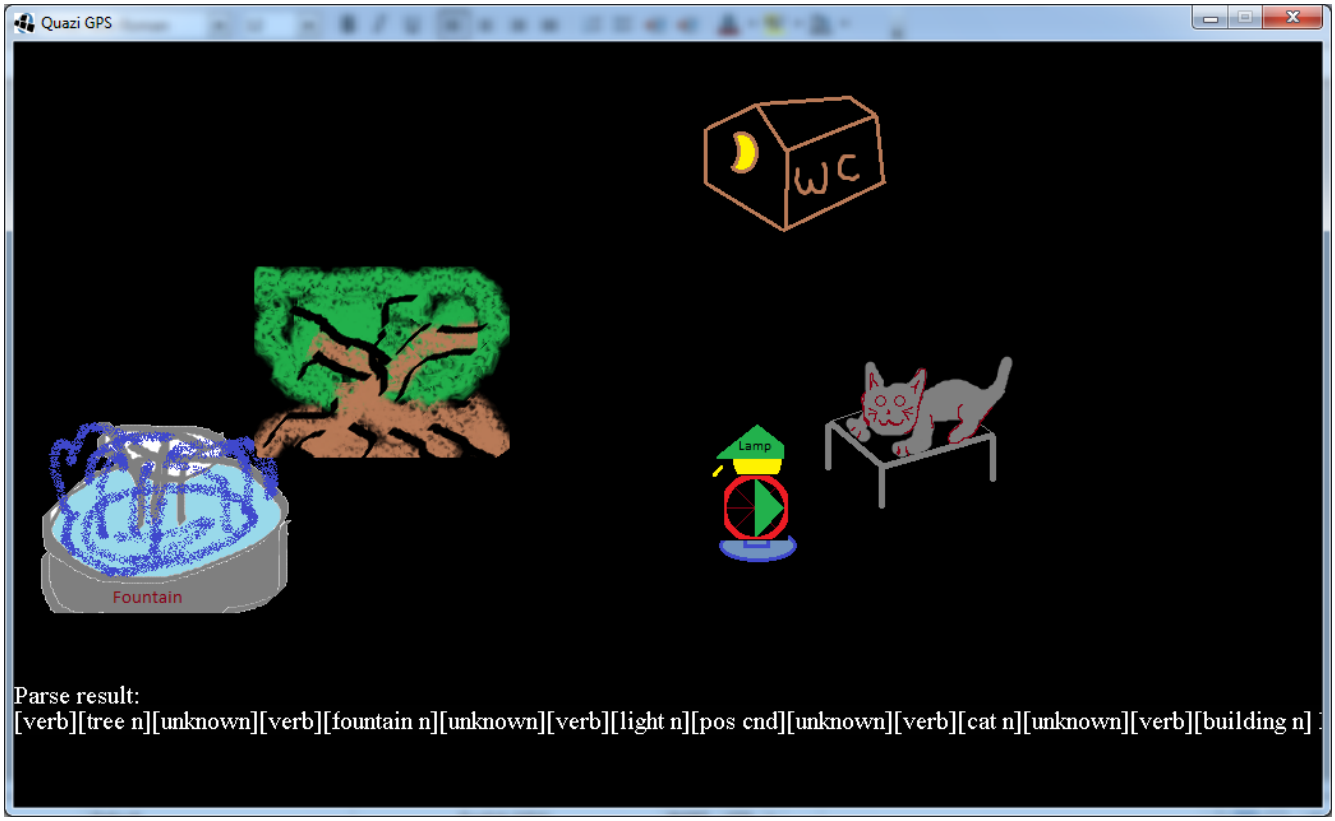
“walk to the tree then run to the fountain next touch the light after that move to the cat finally find the outhouse”

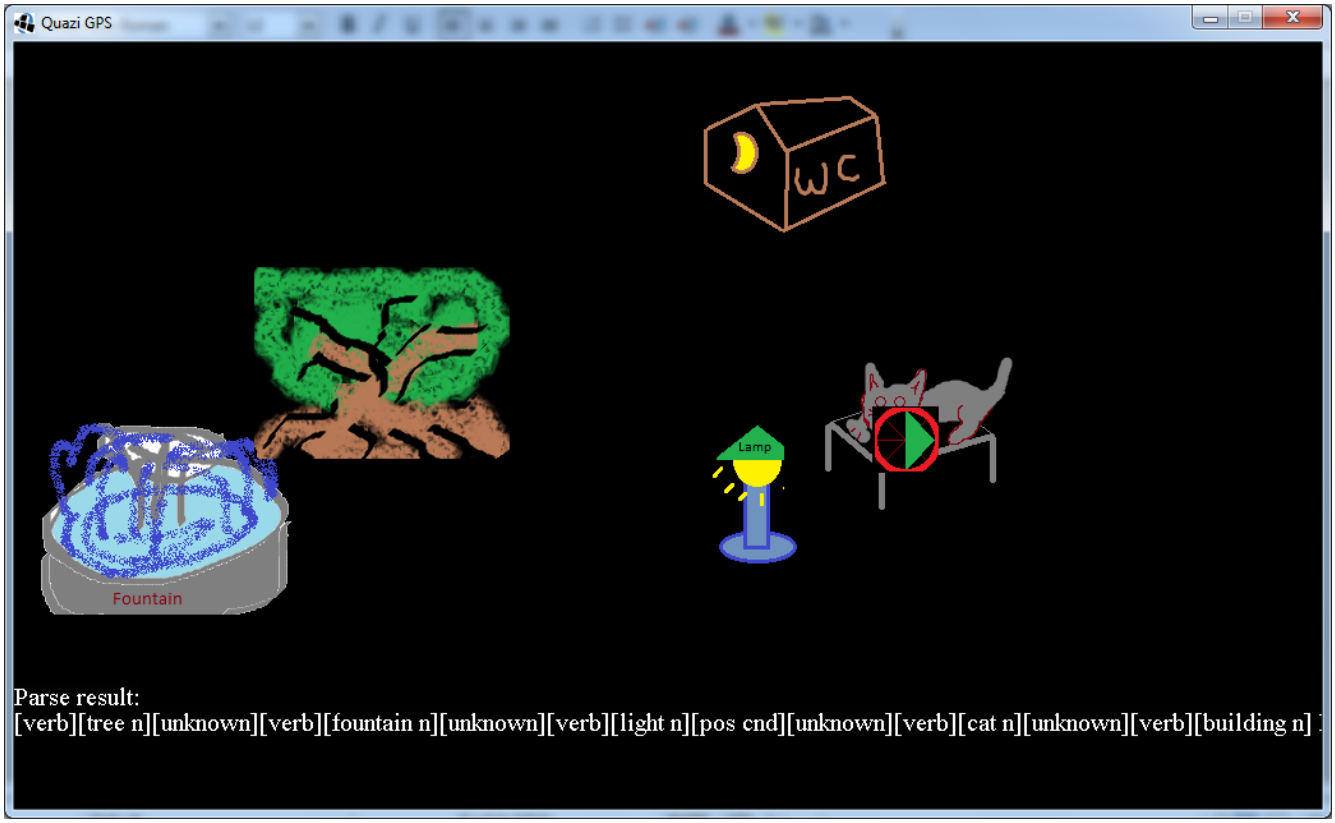

Quazi GPS
WC
Fountain
Type a command, press <Enter> when finished:
walk to the tree then run to the fountain next touch the light after that move to the cat finally find the outhouse

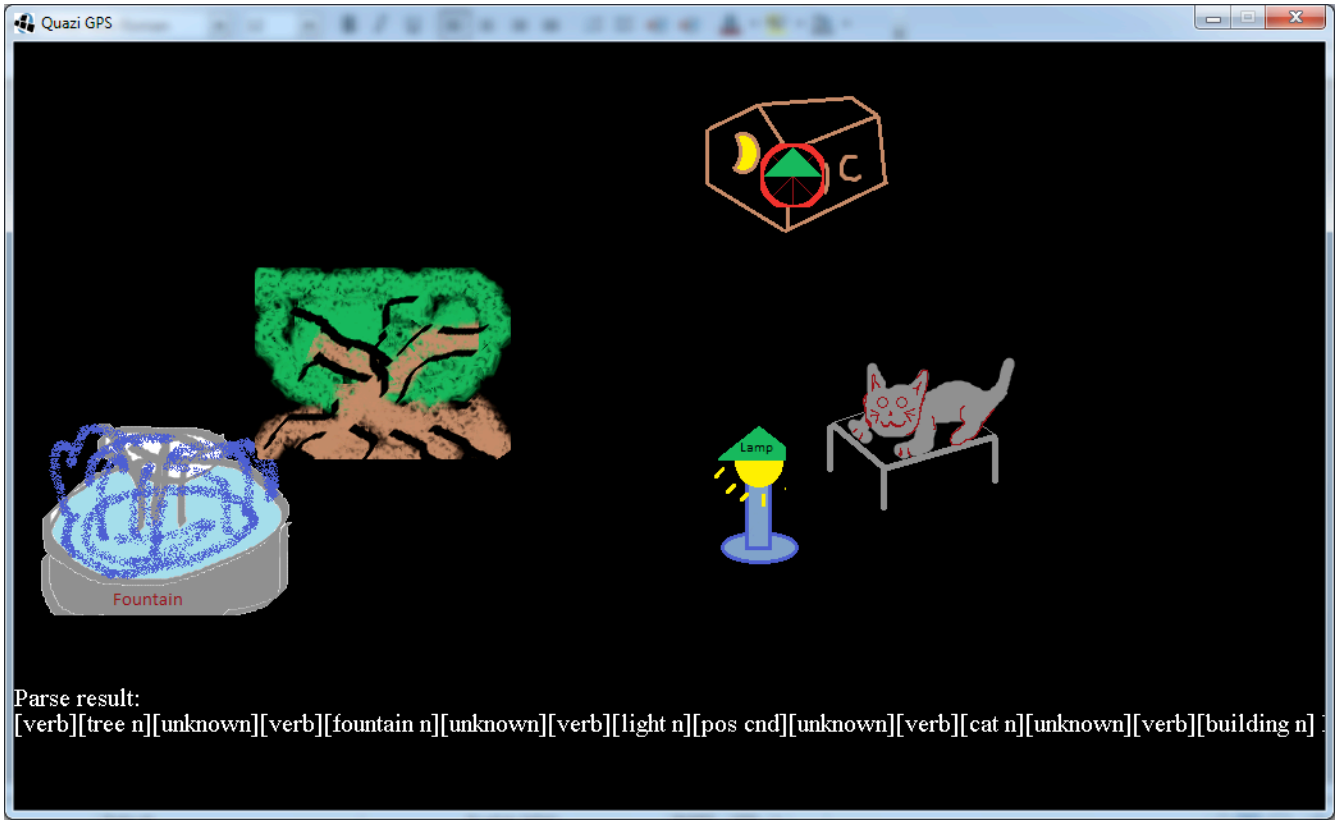

Quazi GPS
WC
Lamp
Fountain
Parse result:
[verb][tree n][unknown][verb][fountain n][unknown][verb][light n][pos cnd][unknown][verb][cat n][unknown][verb][building n]


Quazi GPS
WC
Lamp
Fountain
Parse result:
[verb][tree n][unknown][verb][fountain n][unknown][verb][light n][pos cnd][unknown][verb][cat n][unknown][verb][building n]

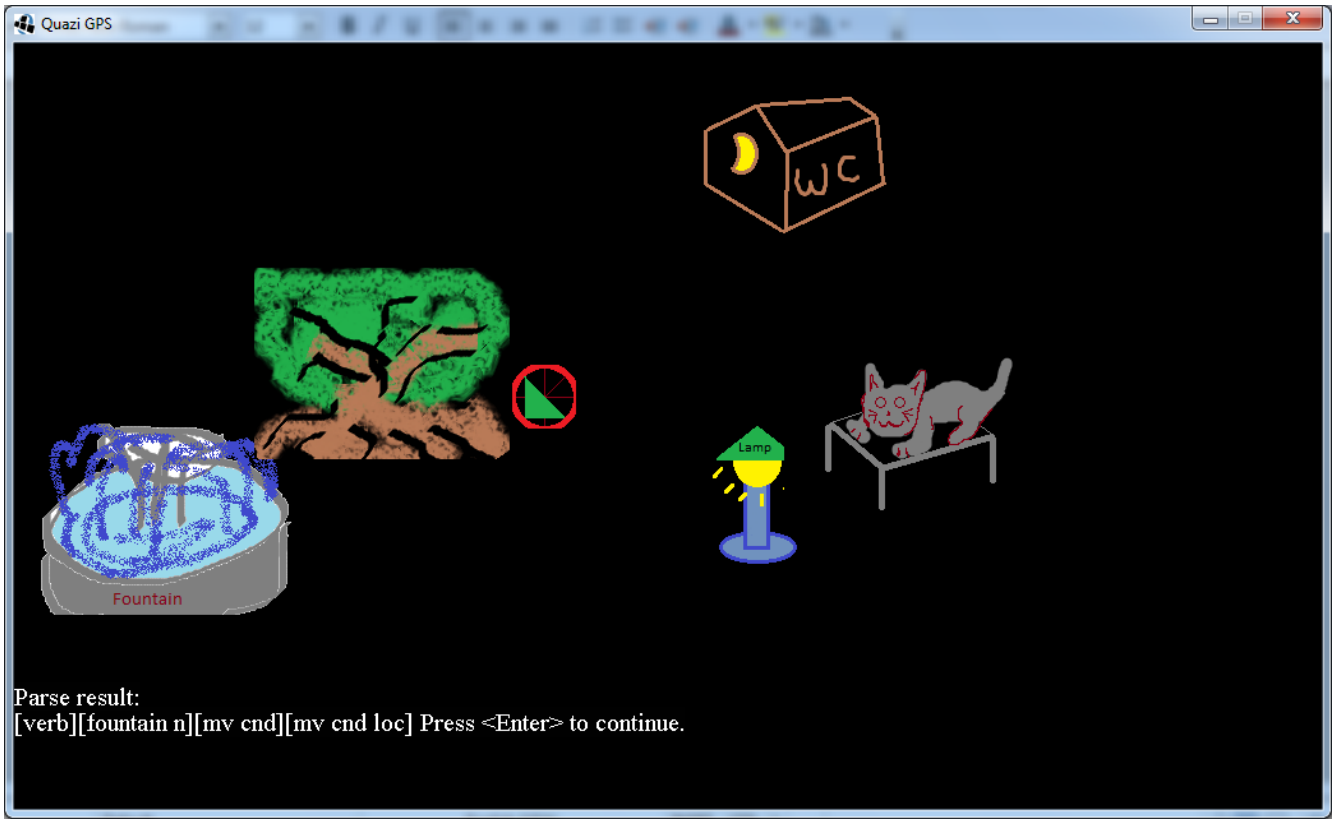

Quazi GPS
WC
Lamp
Fountain
Parse result:
[verb][tree n][unknown][verb][fountain n][unknown][verb][light n][pos cnd][unknown][verb][cat n][unknown][verb][building n]

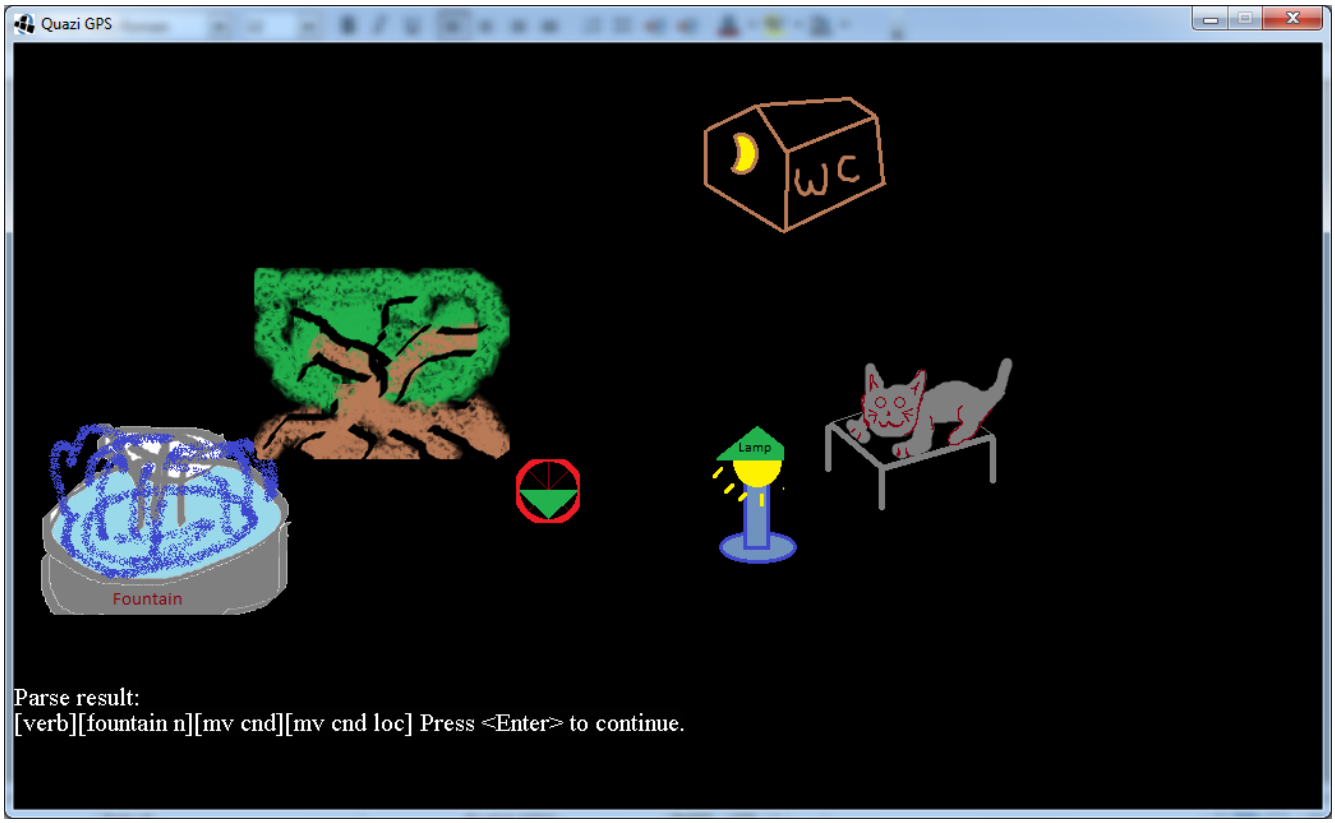

Quazi GPS
WC
Lamp
Fountain
Parse result:
[verb][tree n][unknown][verb][fountain n][unknown][verb][light n][pos cnd][unknown][verb][cat n][unknown][verb][building n]

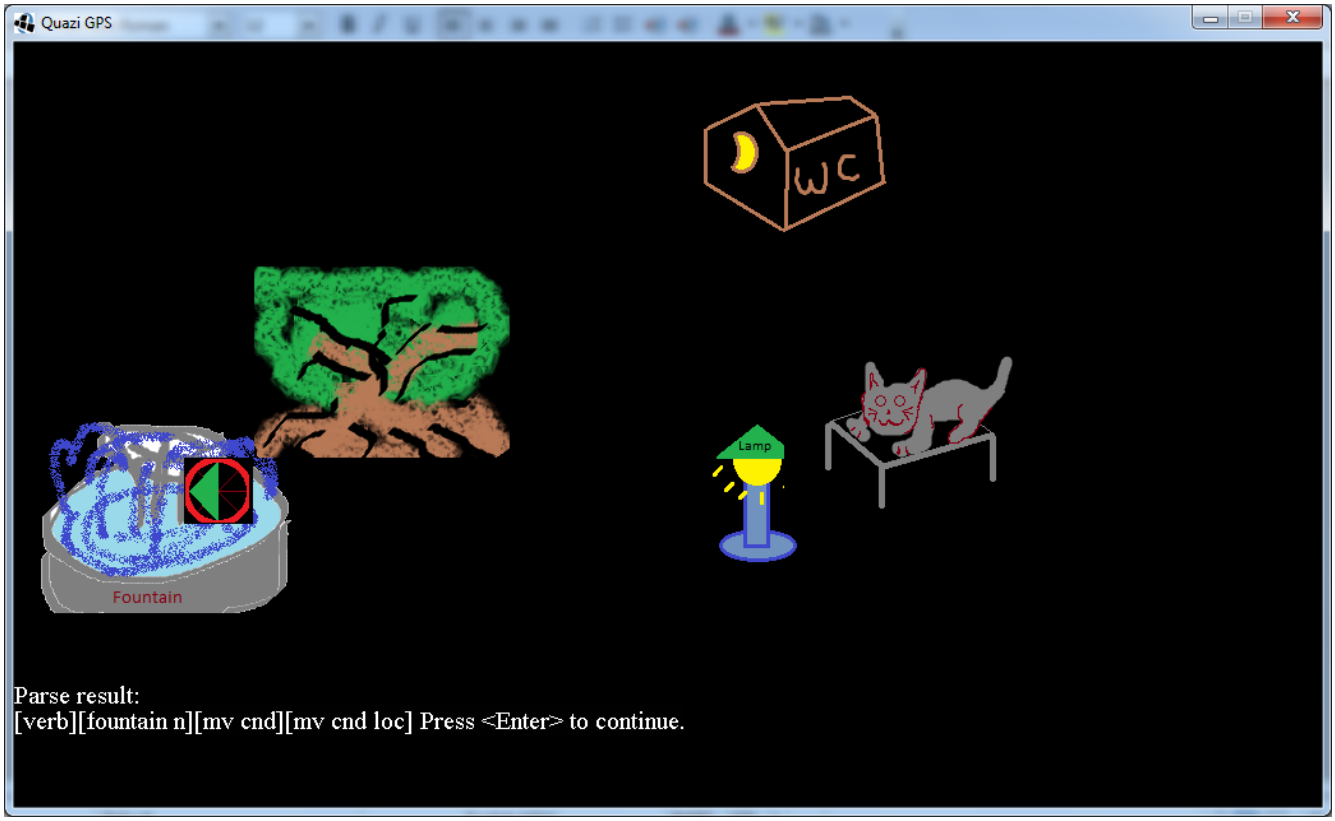

Quazi GPS
WC
Lamp
Fountain
Parse result:
[verb][tree n][unknown][verb][fountain n][unknown][verb][light n][pos cnd][unknown][verb][cat n][unknown][verb][building n]


Quazi GPS
Fountain
Lamp
Parse result:
[verb][tree n][unknown][verb][fountain n][unknown][verb][light n][pos cnd][unknown][verb][cat n][unknown][verb][building n]

“go to the fountain avoid the tree”






Quazi GPS
WC
Lamp
Fountain
Parse result:
[verb][fountain n][mv cnd][mv cnd loc] Press <Enter> to continue.

Code:

Game Class

```
import static org.lwjgl.opengl.GL11.GL_COLOR_BUFFER_BIT;
import static org.lwjgl.opengl.GL11.GL_DEPTH_BUFFER_BIT;
import static org.lwjgl.opengl.GL11.GL_DEPTH_TEST;
import static org.lwjgl.opengl.GL11.GL_MODELVIEW;
import static org.lwjgl.opengl.GL11.GL_PROJECTION;
import static org.lwjgl.opengl.GL11.GL_TEXTURE_2D;
import static org.lwjgl.opengl.GL11.glClear;
import static org.lwjgl.opengl.GL11.glDisable;
import static org.lwjgl.opengl.GL11.glEnable;
import static org.lwjgl.opengl.GL11.glLoadIdentity;
import static org.lwjgl.opengl.GL11.glMatrixMode;
import static org.lwjgl.opengl.GL11.glOrtho;
import static org.lwjgl.opengl.GL11.glViewport;

import java.awt.Font;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Random;

import javax.swing.JOptionPane;

import org.lwjgl.LWJGLException;
import org.lwjgl.Sys;
import org.lwjgl.input.Keyboard;
import org.lwjgl.input.Mouse;
import org.lwjgl.opengl.Display;
import org.lwjgl.opengl.DisplayMode;
import org.lwjgl.opengl.GL11;
import org.newdawn.slick.TrueTypeFont;
import org.newdawn.slick.opengl.TextureImpl;

import com.esotericsoftware.kryonet.Client;
import com.esotericsoftware.kryonet.Connection;
import com.esotericsoftware.kryonet.Listener;
```

```
import com.esotericsoftware.kryonet.Server;


public class Game {
	//necessary LWJGL variables
	String WINDOW_TITLE = "Quazi GPS";
	boolean fullscreen = false;
	int width = Const.wwidth;
	int height = Const.wheight;
	boolean running = true;

	//standard game variables
	int gamestate = 0;
	Input input = new Input();
	TextureLoader textureLoader;
	private TrueTypeFont[] font = new TrueTypeFont[Const.fontAmt];
	static Random gen = new Random();

	//Timing variables
	private static long timerTicksPerSecond = Sys.getTimerResolution();
	long sTime;

	//Game Variables
	Sprite[] assets = new Sprite[Const.artAssetTotal];
	TextArea inputZone = new TextArea(this);
	ArrayList<Entity> entities = new ArrayList<Entity>();

	String debugm = "";




	public Game()throws IOException{
		try {
			setDisplayMode();
			Display.setTitle(WINDOW_TITLE);
			Display.setFullscreen(fullscreen);//always set to false for the moment
			Display.create();
			glEnable(GL_TEXTURE_2D);
			glDisable(GL_DEPTH_TEST);
			glMatrixMode(GL_PROJECTION);
			//something in here enables fonts. Note: fonts still need to be drawn last with the
activator function.
			GL11.glShadeModel(GL11.GL_SMOOTH);
			GL11.glDisable(GL11.GL_LIGHTING);
			glMatrixMode(GL_PROJECTION);
			GL11.glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
		GL11.glClearDepth(1);
```

```
            GL11.glEnable(GL11.GL_BLEND);
            GL11.glBlendFunc(GL11.GL_SRC_ALPHA, GL11.GL_ONE_MINUS_SRC_ALPHA);
                    //-------------------------
                    glLoadIdentity();
                    glOrtho(0, width, height, 0, -1, 1);
                    glMatrixMode(GL_MODELVIEW);
                    glLoadIdentity();
                    glViewport(0, 0, width, height);
                    textureLoader = new TextureLoader();
               } catch (LWJGLException le) {
                    System.out.println("Game exiting - exception in initialization:");
                    le.printStackTrace();
                    return;
               }




               loadArt();
               initEntities();

        }//end constructor


        //-------------------------------------------------------------------------------------------------------
--------
        private boolean setDisplayMode() {
           try {
                    // get modes
                    DisplayMode[] dm = org.lwjgl.util.Display.getAvailableDisplayModes(width,
height, -1, -1, -1, -1, 60, 60);

            org.lwjgl.util.Display.setDisplayMode(dm, new String[] {
               "width=" + width,
               "height=" + height,
               "freq=" + 60,
               "bpp=" + org.lwjgl.opengl.Display.getDisplayMode().getBitsPerPixel()
              });

            if(Const.debug){
                 for(int i = 0; i < dm.length; i++){
                      System.out.println("DM["+ i+ "]" + dm.toString() + ". Resolution(WxH): " +
dm[i].getWidth() + "x" + dm[i].getHeight() + "@" + dm[i].getBitsPerPixel());
                 }
            }

            return true;
           } catch (Exception e) {
```

```
            e.printStackTrace();
            System.out.println("Unable to enter fullscreen, continuing in windowed mode");
          }
                return false;
        }//end setdisplaymode


        //-------------------------------------------------------------------------------------------------------
--------
        private void  loadArt(){
                assets[Const.aTree] = getSprite("assets/tree.png");
                assets[Const.aFountain] = getSprite("assets/fountain.png");
                assets[Const.aCat] = getSprite("assets/cat.png");
                assets[Const.aLight] = getSprite("assets/light.png");
                assets[Const.aBuilding] = getSprite("assets/building.png");
                assets[Const.aUser] = getSprite("assets/userN.png");
                assets[Const.aUser+1] = getSprite("assets/userNE.png");
                assets[Const.aUser+2] = getSprite("assets/userE.png");
                assets[Const.aUser+3] = getSprite("assets/userSE.png");
                assets[Const.aUser+4] = getSprite("assets/userS.png");
                assets[Const.aUser+5] = getSprite("assets/userSW.png");
                assets[Const.aUser+6] = getSprite("assets/userW.png");
                assets[Const.aUser+7] = getSprite("assets/userNW.png");


                //and fonts

                Font awtFont = new Font("Times New Roman", Font.BOLD, 12);
                font[Const.fontDebug] = new TrueTypeFont(awtFont, false);
                awtFont = new Font("Times New Roman", Font.PLAIN, 20);
                font[Const.fontCard] = new TrueTypeFont(awtFont, false);



        }//end load art

        //-------------------------------------------------------------------------------------------------------
--------
        public Sprite getSprite(String ref) {
                return new Sprite(textureLoader, ref);
        }//end getSprite


        //-------------------------------------------------------------------------------------------------------
--------
        private void  initEntities(){

                User temp1 = new User(50,50, Const.userSize,Const.userSize, assets[Const.aUser],
```

```
this);
                entities.add(temp1);
                Entity temp2 = new Entity(200,400, Const.objectW,Const.objectH,
assets[Const.aTree], "tree");
                entities.add(temp2);
                temp2 = new Entity(400,200, Const.objectW,Const.objectH, assets[Const.aFountain],
"fountain");
                entities.add(temp2);
                temp2 = new Entity(400,200, Const.objectW,Const.objectH, assets[Const.aCat], "cat");
                entities.add(temp2);
                temp2 = new Entity(400,200, Const.objectW,Const.objectH, assets[Const.aBuilding],
"building");
                entities.add(temp2);
                temp2 = new Entity(400,200, Const.objectW,Const.objectH, assets[Const.aLight],
"light");
                entities.add(temp2);

                randomize();

        }//end load art

        //randomizes location of objects and user. Clears out user checkpoints
        public void randomize(){
                ((User) entities.get(0)).dCount = 0;
                int x=0;
                int y = 0;

                boolean[] spots = {false, false, false, false, false};
                int j;
                for(int i = 1; i < 6 ; i++){

                        j = (int) ( Math.random() * 5);
                        while( spots[j] == true){
                                j = (int) ( Math.random() * 5);
                        }

                        spots[j] = true;

                        j++;
                        if(i == 1){
                                x = 100;
                                y = 50;
                                x += (int) (Math.random() * 314);
                                y += (int) (Math.random() * 250);
                                entities.get(j).cx = x;
                                entities.get(j).cy = y;
                        }
                        else if(i == 2){
```

```
                        x = 514;
                        y = 50;
                        x += (int) (Math.random() * 314);
                        y += (int) (Math.random() * 250);
                        entities.get(j).cx = x;
                        entities.get(j).cy = y;
                }
                else if(i == 3){
                        x = 100;
                        y = 300;
                        x += (int) (Math.random() * 314);
                        y += (int) (Math.random() * 250);
                        entities.get(j).cx = x;
                        entities.get(j).cy = y;
                }
                else if(i == 4){
                        x = 514;
                        y = 300;
                        x += (int) (Math.random() * 314);
                        y += (int) (Math.random() * 250);
                        entities.get(j).cx = x;
                        entities.get(j).cy = y;
                }
                else if(i == 5){
                        x = 312;
                        y = 150;
                        x += (int) (Math.random() * 400);
                        y += (int) (Math.random() * 300);
                        entities.get(j).cx = x;
                        entities.get(j).cy = y;
                }
        }//end for each obstacle randomize

        x = 0;
        y = 0;
        x += (int) (Math.random() * 1024);
        y += (int) (Math.random() * 600);
        entities.get(0).cx = x;
        entities.get(0).cy = y;


    }//end randomize



    //-------------------------------------------------------------------------------------------------------------
--------
    public void loop(){
```

```
                sTime = getTime();
                while(running){
                        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
                        glMatrixMode(GL_MODELVIEW);
                        glLoadIdentity();

                        switch(gamestate){
                                case Const.gsTitle:
                                {
                                        //show title
                                        drawAssets();
                                        drawFonts();//fonts must be drawn after assets.
                                        inputProcess();
                                        entities.get(0).update();//hard coded, for updating user icon

                                        //gamestate = Const.gsGameLoop;
                                        break;
                                }//end case title

                                case Const.gsGameLoop:
                                {
                                        //gameloop

                                        inputProcess();
                                        //draw function
                                        drawAssets();
                                        drawFonts(); //fonts must be done last


                                        break;
                                }//end case gameloop

                                default:
                                {
                                        break;
                                }
                        }//end switch gamestate
                        Display.update();
                        //System.out.print(".");
                        if (Display.isCloseRequested() ||
Keyboard.isKeyDown(Keyboard.KEY_ESCAPE)) {
                                running = false;
                        }
                }//end while running
                Display.destroy();
        }//end loop

        //-----------------------------------------------------------------------------------------------------
```

```
--------
	//process all input from keyboard
	//this method assumes ready to grab player input
	public void inputProcess(){
		String bval = "";
		char alpha = 'a';
		//dont forget to poll.
		input.poll();


		for(int i = 'a'; i <= 'z'; i++){
			alpha = (char) i;
			bval = alpha + "";
			if(input.isPressed(bval)){
				inputZone.addToInput(alpha);
				System.out.println("attempting to add:" + alpha);
			}
		}//

		if(input.isPressed(Input.space)){
			inputZone.addToInput(' ');
			System.out.println("attempting to add: [space]");
		}

		if(input.isPressed(Input.bkspb)){
			inputZone.delChar();
			System.out.println("attempting to delete");
		}

		if(input.isPressed(Input.entb)){
			inputZone.enterButton();
			System.out.println("attempting to enter");
		}

	}//end input process


	//-----------------------------------------------------------------------------------------------------------
--------
	public void drawAssets(){
		for(int i = entities.size() -1; i >=0; i--){
			entities.get(i).draw();
		}

		if(Const.debug){
		}

	}//
```

```
	//-------------------------------------------------------------------------------------------------------------
--------
	public void drawFonts(){

		TextureImpl.bindNone(); //this is absolutely )$(*@&( necessary to render fonts. Why?
IDK
		String output;

		if(gamestate == Const.gsTitle){

			if(inputZone.isPolling()){
				output = inputZone.getIM();
				font[Const.fontCard].drawString( 0, inputZone.getTextAHS(), output );
				output = inputZone.getUI();
				font[Const.fontCard].drawString( 0, inputZone.getTextAHS() + 20 ,
output );
			}
			else{
				output = "Parse result:";
				font[Const.fontCard].drawString( 0, inputZone.getTextAHS(), output );
				output = inputZone.getPI();
				font[Const.fontCard].drawString( 0, inputZone.getTextAHS() + 20 ,
output );
			}

		}//


	}//end draw fonts



	//-------------------------------------------------------------------------------------------------------------
--------
	public static long getTime(){
		return (Sys.getTime() * 1000) / timerTicksPerSecond;
	}




	//-----------------------------------------------------------------------------------
	public static void main(String[] args) throws IOException{

		Game ng = new Game();

		ng.loop();
```

```
        }//end main
}//end class
```

Entity Class

```
public class Entity {

	//pos relate to upper left, center relates to center
	protected int cx, cy; //center x y
	protected int height, width; //height width
	protected Sprite img; //hold image to draw itself.
	protected String type;

	/**
	 *
	 *
	 * @param xcenter
	 * @param ycenter
	 * @param h
	 * @param w
	 * @param image
	 */

	public Entity(int xcenter, int ycenter, int w, int h, Sprite image, String
description ){
		cx = xcenter;
		cy = ycenter;

		height = h;
		width = w;

		img = image;

		type = description;

	}//end constructor


	//POS relates to upper left corner
	public int getXPos(){return cx - width/2;}
	public int getYPos(){return cy - height/2;}

	//draw based off of position
	public void draw(){
		//
		img.draw(cx -  width/2, cy - height/2);
	}

	//nothing currently in super class because different entities will implement
differently.
	public void update(){};//empty class for implementation by subclasses
```

```
}//end object
```

User Class

```
public class User extends Entity {

	int speed = 3;
	long lastUpdateTime;

	//have bits on figuring out how to move
	//checkpoints? Destination locations?
	int dCount = 0;
	int maxD = 20;
	int[] destX = new int[maxD];//at most, maxD checkpoints
	int[] destY = new int[maxD];
	int facing = 0; //0 = north, 1 = NE, 2 = E, ...6 = w, 7 = nw.

	Game map;


	//constructor
	public User(int xcenter, int ycenter, int h, int w, Sprite image, Game session ){
		super(xcenter, ycenter, h, w, image, "user");
		map = session;

		lastUpdateTime = Game.getTime();

		//hardcoded to test move speeds, and updater fucntion
		addDestination(100,100);
		addDestination(100, Const.wheight-200);




	}//end constructor


	//Update is for the real time movement of the user
	public void update(){
		int movedHorizontal = 0;//0 is no move, -1 = west, +1 = east.
		int movedVertical = 0;// 0 is no move, -1 = north, +1 = south

		//update at roughly 40 fps.
		if(Game.getTime() > lastUpdateTime + 40){
			lastUpdateTime = Game.getTime();
			//if there are destinations
			if(dCount > 0){
				//if there is a difference in current x and destination x
				if(destX[0] != cx){
					int dx = Math.abs(cx - destX[0]);
					if(dx < speed){
						cx = destX[0];
					}
```

```
                                else if(destX[0] > cx){
                                        cx += speed;
                                        movedHorizontal = 1;
                                }
                                else{
                                        cx -= speed;
                                        movedHorizontal = -1;
                                }
                        }//end if dist difference
                        //if there is a difference in current y and destination y
                        if(destY[0] != cy){
                                int dy = Math.abs(cy - destY[0]);
                                if(dy < speed){
                                        cy = destY[0];
                                }
                                else if(destY[0] > cy){
                                        cy += speed;
                                        movedVertical = 1;
                                }
                                else{
                                        cy -= speed;
                                        movedVertical = -1;
                                }
                        }//end if y pos doesn't match

                        //if the destinations match, remove that destination point
                        if(cx == destX[0] && cy == destY[0]){
                                dCount--;
                                for(int i = 0; i < dCount; i++){
                                        destX[i] = destX[i+1];
                                        destY[i] = destY[i+1];
                                }
                        }//end destinatin point update

                }//end if there are destination

        }//end 1/10 sec update

        //if there was horizontal movement
        int tempFace = 0;
        if(movedHorizontal != 0){
                tempFace = 2*(2 - movedHorizontal);//left is -1, right + 1. Direction
face 2 = east, 6 = west.
                //if moved north
                if(movedVertical == -1 && tempFace == 6) tempFace += 1;
                else if(movedVertical == -1 && tempFace == 2) tempFace -= 1;
                else if(movedVertical == 1 && tempFace == 6) tempFace -= 1;
                else if(movedVertical == 1 && tempFace == 2) tempFace += 1;

                facing = tempFace;
        }
        else if(movedVertical != 0){
                tempFace = 2*(1 + movedVertical);//up is -1, down + 1. Direction face
0 = north, 4 = south.
                facing = tempFace;
        }
```

```
	}//end update

	public void addDestination(int x, int y){
		if(dCount < maxD){
		destX[dCount] = x;
		destY[dCount] = y;
		dCount++;
		}
	}//end addDestination


	/**get modifiers for facing value.
	 *if facing:
	 *N: (0,-1) since monitor origin is in top left, NE:(1,-1), etc
	 *
	 *
	 * @param xy array that will be changed
	 * @param afterMod a direction to turn. 0:forward, 1:right, 2:back,3:left
	 */
	//*/
	public void getFaceMod(int[] xy, int afterMod){
		int nface = facing;
		if(afterMod == 3){
			//if turning left.
			nface -= 2;
			if(nface < 0){
				nface += 8;
			}
		}
		if(afterMod == 1){
			//turning right
			nface += 2;
			if(nface > 7){
				nface -= 8;
			}
		}
		if(afterMod == 2){
			//backward
			nface += 4;
			if(nface > 7){
				nface -= 8;
			}
		}
		switch(nface){
		case 0:
			xy[0] = 0;
			xy[1] = -1;
			break;
		case 1:
			xy[0] = 1;
			xy[1] = -1;
			break;
		case 2:
			xy[0] = 1;
			xy[1] = 0;
			break;
		case 3:
```

```
                        xy[0] = 1;
                        xy[1] = 1;
                        break;
                case 4:
                        xy[0] = 0;
                        xy[1] = 1;
                        break;
                case 5:
                        xy[0] = -1;
                        xy[1] = 1;
                        break;
                case 6:
                        xy[0] = -1;
                        xy[1] = 0;
                        break;
                case 7:
                        xy[0] = -1;
                        xy[1] = -1;
                        break;
                }

        }//end getfacing


        //draw image based off of position
        public void draw(){
                //
                map.assets[Const.aUser + facing].draw(cx -  width/2, cy - height/2);
        }


}//end class
```

Input Class

```
import org.lwjgl.input.Keyboard;
import org.lwjgl.input.Mouse;




//-----------------------------------------------------------------------------------------------------------------------
public class Input {

        final int alphaAmt = 26;//how many alphas are being tracked. 26 for standard english.
        final int mbStart = alphaAmt; //index position of mouse button starts
        final int mbAmt = 2; //MouseButton Amount
        final int entBkspStart = alphaAmt + mbAmt;
        final int entBkspAmt = 3; //enter, backspace, space
        final int totalIndex = alphaAmt + mbAmt + entBkspAmt;
```

```
	//The following variables will track button states.
	boolean[] pressed = new boolean[totalIndex];
	boolean[] released = new boolean[totalIndex];
	boolean[] down = new boolean[totalIndex];

	//string notations for buttongs
	static final String mlb = "mlb";
	static final String mrb = "mrb";
	static final String entb = "entb";
	static final String bkspb = "bkspb";
	static final String space = "space";


	//-------------------------------------------------------------------------------------------------------------------
--------
	public Input(){
		//for each check value
		for(int i = 0; i < totalIndex; i++){
			pressed[i] = false;
			released[i] = false;
			down[i] = false;
		}//



		System.out.println("Debug: what is index of a, enter and backspace?");
		System.out.println("a:" + getIndex("a"));
		System.out.println("enter:" + getIndex(entb));
		System.out.println("back space:" + getIndex(bkspb));

	}//end constructor


	//-------------------------------------------------------------------------------------------------------------------
--------
	//poll function will update the status of the button states.
	public void poll(){

		char cap = 'A';
		int index = 0;
		String val = "";

		//SLOWSPOT: this polls all the alphas on the keyboard.
		//not all games need all alphas checked.
		while(cap <= 'Z'){
			val = "" + cap;
			//if key is down
			if(Keyboard.isKeyDown(Keyboard.getKeyIndex(val))){
```

```
                                //if they key was not registered as already being down
                                if(!down[index]){
                                        pressed[index] = true;//then it was just pressed
                                        down[index] = true; //and it is now down
                                        released[index] = false;//safety
                                        //debug
                                        if(Const.debug){
                                        System.out.println(val + " was pressed");
                                        }
                                }//
                                else{
                                        //if it was already down, then pressed needs to change to false.
                                        //Note: there is not if statement, because an if statement is slower
than just assigning a value.
                                        pressed[index] = false;
                                }
                        }//end is A key down
                        else{
                                //if the key is not down, but is listed as down...
                                if(down[index]){
                                        down[index] = false; //change to not down
                                        pressed[index] = false; //just in case, pressed goes false
                                        released[index] = true; //key was just released
                                        if(Const.debug){
                                        System.out.println(val + " was released");
                                        }
                                }//
                                else{
                                        released[index] = false; //if it wasn't previously down, then
released is false.
                                }
                        }//end key was not down

                        cap++;
                        index++;

                }//end the keystate checks for alphas.


                //now check for mouse button states.
                for(int i = 0; i < mbAmt; i++){
                        if(Mouse.isButtonDown(i)){
                                //if they key was not registered as already being down
                                if(!down[index]){
                                        pressed[index] = true;//then it was just pressed
                                        down[index] = true; //and it is now down
                                        released[index] = false;
                                        //debug
```

```
                                        if(Const.debug){
                                        System.out.println("Mouse button " + i + " was pressed");
                                        }
                                }//
                                else{
                                        //if it was already down, then pressed needs to change to false.
                                        //Note: there is not if statement, because an if statement is slower
than just assigning a value.
                                        pressed[index] = false;
                                }
                        }//end is A key down
                        else{
                                //if the key is not down, but is listed as down...
                                if(down[index]){
                                        down[index] = false; //change to not down
                                        pressed[index] = false; //just in case, pressed goes false
                                        released[index] = true; //key was just released
                                        if(Const.debug){
                                        System.out.println("Mouse button " + i + " was released");
                                        }
                                }//
                                else{
                                        released[index] = false; //if it wasn't previously down, then
released is false.
                                }
                        }//end key was not down

                        index++;
                }//end for each mouse button poll

                //Note to self: try break these function down further.
                //adding individual keys later will be a major pain
                //-------------------------------------------------------------------------------------------------
                //now check for enter and backspace button states.
                if(Keyboard.isKeyDown(Keyboard.KEY_RETURN)){
                        //if they key was not registered as already being down
                        if(!down[index]){
                                pressed[index] = true;//then it was just pressed
                                down[index] = true; //and it is now down
                                released[index] = false;
                                //debug
                                if(Const.debug){
                                System.out.println("Enter button was pressed");
                                }
                        }//
                        else{
                                //if it was already down, then pressed needs to change to false.
                                //Note: there is not if statement, because an if statement is slower than
```

```
just assigning a value.
                                pressed[index] = false;
                        }
                }//end is key down
                else{
                        //if the key is not down, but is listed as down...
                        if(down[index]){
                                down[index] = false; //change to not down
                                pressed[index] = false; //just in case, pressed goes false
                                released[index] = true; //key was just released
                                if(Const.debug){
                                System.out.println("Enter button was released");
                                }
                        }//
                        else{
                                released[index] = false; //if it wasn't previously down, then released is
false.
                        }
                }//enter key was not down
                index++;
                ///
                ///Check for backspace now
                ///
                //now check for enter and backspace button states.
                if(Keyboard.isKeyDown(Keyboard.KEY_BACK)){
                        //if they key was not registered as already being down
                        if(!down[index]){
                                pressed[index] = true;//then it was just pressed
                                down[index] = true; //and it is now down
                                released[index] = false;
                                //debug
                                if(Const.debug){
                                System.out.println("Backspace button was pressed");
                                }
                        }//
                        else{
                                //if it was already down, then pressed needs to change to false.
                                //Note: there is not if statement, because an if statement is slower than
just assigning a value.
                                pressed[index] = false;
                        }
                }//end is key down
                else{
                        //if the key is not down, but is listed as down...
                        if(down[index]){
                                down[index] = false; //change to not down
                                pressed[index] = false; //just in case, pressed goes false
                                released[index] = true; //key was just released
```

```
                        if(Const.debug){
                        System.out.println("Backspace button was released");
                        }
                }//
                else{
                        released[index] = false; //if it wasn't previously down, then released is
false.
                }
        }//enter key was not down
        index++;
        ///
        ///Check for space now
        ///
        //now check for enter and backspace button states.
        if(Keyboard.isKeyDown(Keyboard.KEY_SPACE)){
                //if they key was not registered as already being down
                if(!down[index]){
                        pressed[index] = true;//then it was just pressed
                        down[index] = true; //and it is now down
                        released[index] = false;
                        //debug
                        if(Const.debug){
                        System.out.println("Space button was pressed");
                        }
                }//
                else{
                        //if it was already down, then pressed needs to change to false.
                        //Note: there is not if statement, because an if statement is slower than
just assigning a value.
                        pressed[index] = false;
                }
        }//end is key down
        else{
                //if the key is not down, but is listed as down...
                if(down[index]){
                        down[index] = false; //change to not down
                        pressed[index] = false; //just in case, pressed goes false
                        released[index] = true; //key was just released
                        if(Const.debug){
                        System.out.println("Space button was released");
                        }
                }//
                else{
                        released[index] = false; //if it wasn't previously down, then released is
false.
                }
        }//enter key was not down
        index++;
```

```
	}//end poll

	//-------------------------------------------------------------------------------------------------------------
--------
	public boolean isDown(String val){
		int pos = getIndex(val);

		if(pos != -1){

			return down[pos];

		}

		return false;
	}//end isDown

	//-------------------------------------------------------------------------------------------------------------
--------
	public boolean isPressed(String val){
		int pos = getIndex(val);

		if(pos != -1){

			return pressed[pos];

		}

		return false;
	}//end isPressed

	//-------------------------------------------------------------------------------------------------------------
--------
	public boolean isReleased(String val){
		int pos = getIndex(val);

		if(pos != -1){

			return released[pos];

		}

		return false;
	}//end isreleased
```

```
	//-------------------------------------------------------------------------------------------------------------
--------
	//returns the index of a value being tracked.
	public int getIndex(String val){

		int retval = 0;

		if(val.length() == 1){
			//either a number, letter, or symbol.
			char alpha = 'a';
			char nval = val.charAt(0);

			//only do a check if length ==1
			if(val.length() == 1){

				while(alpha != nval){
					alpha++;
					retval++;

					if(alpha > 'z'){
						retval = -1;
						break; //if it is not an alpha, then this will break one
character after lower case 'z'
					}
				}//
			}//end if length 1
			else retval = -1; //no match for the alphas

		}//if one character in size


		if(val.contentEquals(mlb)){
			retval =  mbStart;
		}//if mouse left button
		else if(val.contentEquals(mrb)){
			retval = mbStart + 1;
		}//if mouse left button
		else if(val.contentEquals(entb)){
			retval = entBkspStart;
		}//if mouse left button
		else if(val.contentEquals(bkspb)){
			retval = entBkspStart + 1;
		}//if mouse left button
		else if(val.contentEquals(space)){
			retval = entBkspStart + 2;
		}//if mouse left button
```

```
                return retval; // -1 is an error
        }//end getindex


}//end input class
```

TextArea Class

```
import java.awt.Rectangle;
import java.awt.geom.Line2D;


public class TextArea {

        private String inputMarker = "Type a command, press <Enter> when finished:";
        private String parseInfo = "";
        private String userInput = "";

        private Boolean gettingInput = true;

        private int workAreaH = Const.wheight - Const.textArea;
        private int workAreaW = Const.wwidth;

        private Game session;


        //Vocabulary
        //Nouns
        private String[] nounList = { "tree", "bush","plant", "oak", "brush", "green", "greens",
"fountain", "water","sprinkler", "sprayer"};//note this is a manual update
        private String[] treeList= { "tree", "bush","plant", "oak", "brush", "green", "greens"};
        private String[] fountainList= { "fountain", "water","sprinkler", "sprayer"};
        private String[] catList= { "cat", "statue","furball", "animal", "kitty"};
        private String[] lightList= { "light", "lamp","torch", "pole", "bulb"};
        private String[] buildingList= { "building", "structure","outhouse", "bathroom", "facilities",
"wc", "house"};

        //other
        private String[] stopList= { "stop", "halt","freeze", "until"};
        private String[] verbList= { "go", "walk","move","run","travel", "turn", "touch", "find" };
        private String[] moveCondList= { "dont", "around","avoid","away" }; //iffy on travel and turn
        private String[] posCondList= { "past", "in", "above", "below", "after", "beyond", "before",
"top", "bottom" };
        private String[] fluffList= { "the", "a", "to", "toward", "from", "of", "ahead", "and", "side",
"by" };
        private String[] nounDirList= { "left", "infront", "front", "right", "forward", "forwards",
"back", "backwards", "backward", "behind" };
        private String[] mapDirList= { "up", "down", "north", "south", "east", "west", "northeast",
```

```
"northwest", "southeast", "southwest" };


	private String[][] vocabList = {verbList, moveCondList, posCondList, fluffList, nounDirList,
mapDirList};


	//constructor get a reference to session.
	public TextArea(Game environment){
		session = environment;



	}

	//add a character to the current input string
	public void addToInput(char nextc){
		if(gettingInput){
			userInput = userInput + nextc;
		}
	}

	//delete the last char of the input string
	public void delChar(){
		if(userInput.length() > 0) {userInput = userInput.substring(0, userInput.length() - 1);}
	}

	//enter button was pressed, either parse string or clear the parse info from pane
	public void enterButton(){
		if(gettingInput){
			if(userInput.contentEquals("reset")){
				resetInput();
			}
			else if(userInput.contentEquals("barrelroll") ||
userInput.contentEquals("barelroll")|| userInput.contentEquals("barellroll")||
userInput.contentEquals("barrellroll")){
				barrelRoll();
			}
			else if(userInput.split(" ").length <= 2){
				parseHalt();
			}
			else{
				parseString();
			}
			gettingInput = false;
			userInput = "";
		}
		else{
```

```
                gettingInput = true;
        }
}


//resets the board.
public void resetInput(){
        //need to fill this out later.
        session.randomize();
}


//Attempts a barrel roll
public void barrelRoll(){
        parseInfo = "Do a barrel roll!";

        int x = session.entities.get(0).cx;
        int y = session.entities.get(0).cy;
        int d = 4;

        ((User)session.entities.get(0)).dCount = 0;//remove checkpoints.

        //go up
        y -= d;
        ((User)session.entities.get(0)).addDestination(x, y);
        //go up left
        y -= d;
        x -= d;
        ((User)session.entities.get(0)).addDestination(x, y);
        //go left
        x -= d;
        ((User)session.entities.get(0)).addDestination(x, y);
        //go down left
        y += d;
        x -= d;
        ((User)session.entities.get(0)).addDestination(x, y);
        //go down
        y += d;
        ((User)session.entities.get(0)).addDestination(x, y);
        //go down right
        y += d;
        x += d;
        ((User)session.entities.get(0)).addDestination(x, y);
        //go  right
        x += d;
        ((User)session.entities.get(0)).addDestination(x, y);
        //go up right
        y -= d;
```

```
            x += d;
            ((User)session.entities.get(0)).addDestination(x, y);
            //go up
            y -= d;
            ((User)session.entities.get(0)).addDestination(x, y);
            //go up left
            y -= d;
            x -= d;
            ((User)session.entities.get(0)).addDestination(x, y);
            //go left
            x -= d;
            ((User)session.entities.get(0)).addDestination(x, y);
            //go down left
            y += d;
            x -= d;
            ((User)session.entities.get(0)).addDestination(x, y);
            //go down
            y += d;
            ((User)session.entities.get(0)).addDestination(x, y);
            //go down right
            y += d;
            x += d;
            ((User)session.entities.get(0)).addDestination(x, y);
            //go  right
            x += d;
            ((User)session.entities.get(0)).addDestination(x, y);
            //go up right
            y -= d;
            x += d;
            ((User)session.entities.get(0)).addDestination(x, y);
            //go up
            y -= d;
            ((User)session.entities.get(0)).addDestination(x, y);


    }//end barrelRoll


    //parses a halt command
    //on halt a user will stop at current location and empty out all checkpoints.
    public void parseHalt(){

            if(userInput.contains("halt")){
                    parseInfo = "Keyword: [halt]";
                    ((User) session.entities.get(0)).dCount = 0;
            }
            else if(userInput.contains("stop")){
                    parseInfo = "Keyword: [halt]";
```

```
                        ((User) session.entities.get(0)).dCount = 0;
                }
                else if(userInput.contains("freeze")){
                        parseInfo = "Keyword: [halt]";
                        ((User) session.entities.get(0)).dCount = 0;
                }
                else{
                        String[] words = userInput.split(" ");
                        boolean hasVerb = false;
                        boolean hasDirection = false;
                        if(words.length ==2){
                                if(isType(verbList, words[0])){
                                        parseInfo = "[verb][direction]";
                                        if(isType(nounDirList, words[1])){
                                                //((User)session.entities.get(0)).addDestination(x,
y)
                                                //need method to express go in direction until hit
wall.
                                                //IMPLEMENT: see if the move in dir until
method works.
                                                //private String[] nounDirList= { "left", "right",
"forward","forwards", "back", "backwards", "backward" };
                                                int[] xy = new int[2];
                                                if(words[1].contentEquals("forward") ||
words[1].contentEquals("forwards") ){

        ((User)session.entities.get(0)).getFaceMod(xy, 0);
                                                }
                                                else if(words[1].contentEquals("left")){

        ((User)session.entities.get(0)).getFaceMod(xy, 3);//3 is left
                                                }
                                                else if(words[1].contentEquals("right") ){

        ((User)session.entities.get(0)).getFaceMod(xy, 1); //1 is left
                                                }
                                                else{

        ((User)session.entities.get(0)).getFaceMod(xy, 2); //2 is backwards(reverse)
                                                }
                                                this.setMoveInDirUntil(xy[0], xy[1], 9999, 9999);
//keep moving in one direciton , screen will time out move.
                                        }
                                        else if(isType(mapDirList, words[1])){
                                                //private String[] mapDirList= { "up", "down",
"north", "south", "east", "west" };
                                                int xmod, ymod;
                                                xmod = ymod = 0;
```

```
                                                if(words[1].contains("south") ||
words[1].contains("down") ){
                                                        ymod = 1;
                                                }
                                                else if(words[1].contains("north") ||
words[1].contains("up") ){
                                                        ymod = -1;
                                                }

                                                if(words[1].contains("east") ){
                                                        xmod = 1;
                                                }
                                                else if(words[1].contains("west") ){
                                                        xmod = -1;
                                                }
                                                        this.setMoveInDirUntil(xmod, ymod, 9999, 9999);
                                                }//end else map dir word
                                                else{
                                                        parseInfo = "Invalid structure. Missing Noun";
                                                }
                                        }//end first word is a verb
                                        else{
                                                parseInfo = "Invalid structure. Missing Verb";
                                        }
                                }//end if length == 2
                                else{
                                        parseInfo = "Unknown Keywords: " + userInput + ". Possible
incomplete sentence.";
                                }
                        }

        }//end parse halt

        //parse the input string
        public void parseString(){

                String[] commands = userInput.split(" ");
                String[] cparse = new String[commands.length];
                int verb = 0;

                //commands = detectCommands(); //discontinued due to way it works.


                //for each word, detect what type of word it is, then associate to a restriction.
                for(int i = 0; i < commands.length; i++){
                        if(isType(stopList, commands[i])) cparse[i] = "stop";
                        else if(isType(verbList, commands[i])){ cparse[i] = "verb"; verb++; }
                        else if(isType(moveCondList, commands[i])) cparse[i] = "mv cnd";
```

```
            else if(isType(posCondList, commands[i])) cparse[i] = "pos cnd";
            else if(isType(fluffList, commands[i])) cparse[i] = "";
            else if(isType(nounDirList, commands[i])) cparse[i] = "n dir";
            else if(isType(mapDirList, commands[i])) cparse[i] = "m dir";
            else if(isType(treeList, commands[i])) cparse[i] = "tree n";
            else if(isType(fountainList, commands[i])) cparse[i] = "fountain n";
            else if(isType(catList, commands[i])) cparse[i] = "cat n";
            else if(isType(lightList, commands[i])) cparse[i] = "light n";
            else if(isType(buildingList, commands[i])) cparse[i] = "building n";
            else cparse[i] = "unknown";
    }//end for each word


    //get my counts
    int donts = 0;
    int stops = 0;
    int verbs = 0;
    int directionOf = 0;
    int moveInDir = 0;
    int[] nouns = new int[20];
    int nounpos = 0;


    //strip down the in line duplicates.
    //example: go[v] walk[v] to the tree[n] => [v] [n]
    for(int i = 0; i < cparse.length; i++){

            //singleton checks
            if(cparse[i].contentEquals("mv cnd")){donts++;}
            else if(cparse[i].contentEquals("stops")){stops++;}
            else if(cparse[i].contentEquals("verb")){verbs++;}
            else if((nounpos < 20) &&
                            ( cparse[i].contentEquals("tree n")
                                            || cparse[i].contentEquals("fountain n")
                                            || cparse[i].contentEquals("cat n")
                                            || cparse[i].contentEquals("light n")
                                            || cparse[i].contentEquals("building n"))){
                            nouns[nounpos] = i;
                            nounpos++;
            }
            //duplicates
            if(i>1){
                    if(cparse[i].contentEquals("mv cnd")){

                            if(i  < cparse.length -1  ){
                                    if(!cparse[i+1].contentEquals("mv cnd")){
                                            for(int j = i + 1; j < cparse.length;j++){
                                                    if(cparse[j].contentEquals("tree n")
```

```
                                                                    ||
cparse[j].contentEquals("fountain n")
                                                                    ||
cparse[j].contentEquals("cat n")
                                                                    ||
cparse[j].contentEquals("light n")
                                                                    ||
cparse[j].contentEquals("building n") ){
                                                                        cparse[j] = "mv cnd loc";
                                                                }
                                                            }
                                                        }
                                                    }
                                    }//end if there was a mv cmd


                                    if(cparse[i].contentEquals("verb") && cparse[i -
1].contentEquals("verb") ) {
                                            cparse[i-1] = "";
                                            verbs--;
                                    }
                                    else if(cparse[i].contentEquals("stop") && cparse[i -
1].contentEquals("stop") ) {
                                            cparse[i-1] = "";
                                            stops--;
                                    }
                                    else if(cparse[i].contentEquals("mv cnd") && cparse[i -
1].contentEquals("mv cnd") ) {
                                            cparse[i - 1] = "";
                                            donts--;
                                    }
                                    else if(cparse[i].contentEquals("verb") && cparse[i -
1].contentEquals("mv cnd") ) {
                                            cparse[i] = "";
                                            verbs--;
                                    }
                                    else if( (cparse[i].contentEquals("of") && cparse[i -1].contentEquals("n
dir"))
                                                    ||      (cparse[i].contentEquals("of") && cparse[i -
1].contentEquals("m dir")) ) {
                                            directionOf++;
                                            moveInDir--;
                                    }
                                    else if( cparse[i].contentEquals("n dir") || cparse[i].contentEquals("n
dir")){
                                            moveInDir++;
                                    }
                            }
```

```
		}//end duplicate reduction


		parseInfo = "";

		//actual parse now
		if(verbs > 0){
			if(nounpos > 0){

				int prevnoun = 0;
				boolean match = false;
				//for each of the found nouns attempt a parse.
				for(int j = 0; j < nounpos; j++){
					match = false;
					//first noun. get position of that noun reduce by 1.
					//so long as it doesn't go before the previous noun or start of
sentence, body code
					for(int i = nouns[j] -1 ; i >= prevnoun; i--){

						//figure out which type of noun this is.
						int nounType = 0; //0:user, 1:
						if(isType(treeList, commands[nouns[j]])) nounType =
Const.aTree;
						if(isType(fountainList, commands[nouns[j]])) nounType
= Const.aFountain;
						if(isType(catList, commands[nouns[j]])) nounType =
Const.aCat;
						if(isType(lightList, commands[nouns[j]])) nounType =
Const.aLight;
						if(isType(buildingList, commands[nouns[j]])) nounType
= Const.aBuilding;
						//more types
						//get position of that noun.
						int destNounX = session.entities.get(nounType).cx;
						int destNounY = session.entities.get(nounType).cy;

						if(cparse[i].contentEquals("verb")){


							//if there are donts. look left then right, stay within
previous noun(or beginning of sentence) and before next verb
							if(donts > 0){

								this.findMvCnd(cparse, commands,
nouns[j], prevnoun, session.entities.get(0).cx, session.entities.get(0).cy, destNounX, destNounY);

								donts--;
							}
```

```
                                                //move to the noun without restrictions

        ((User)session.entities.get(0)).addDestination(destNounX, destNounY);

                                                match = true;
                                                i = prevnoun -1;//out of current noun conditions
detection

                                                if(Const.debug){
                                                        System.out.println("Verb only parse");
                                                        System.out.println("User x,y: " +
((User)session.entities.get(0)).cx + "," + ((User)session.entities.get(0)).cy);
                                                        System.out.println("Tree x,y: " +
session.entities.get(1).cx + "," + session.entities.get(1).cy);

                                                }

                                        }//found a verb meaning: [verb] [noun]
                                        else if(cparse[i].contentEquals("pos cnd")
                                                || (cparse[i-1].contentEquals("n dir") &&
(commands[i].contentEquals("of")    || commands[i].contentEquals("side")))
                                                || (cparse[i-1].contentEquals("m dir") &&
(commands[i].contentEquals("of")    || commands[i].contentEquals("side")))        ){
                                                //found an ending position
                                                //modify dest based upon position type
                                                //private String[] posCondList= { "past",
"touch","in", "above", "below", "after", "beyond", "before" };
                                                //private String[] nounDirList= { "left", "infront",
"front", "right", "forward", "forwards", "back", "backwards", "backward" };
                                                //private String[] mapDirList= { "up", "down",
"north", "south", "east", "west", "northeast", "northwest", "southeast", "southwest" };

                                                //in will have no effect since that is the normal
destination
                                                //first up is map position, as it is easiest. Note
north, south, east, west, marked by CONTAINS because it might be northeast or similar
                                                //north
                                                if(       commands[i-1].contentEquals("up")
        ||commands[i-1].contains("north")    ||commands[i].contentEquals("above") ||commands[i-
1].contentEquals("top")){destNounY -= (Const.objectH/2 + Const.userSize/2);}
                                                //south
                                                else if( commands[i-1].contentEquals("down")
        ||commands[i-1].contains("south")    ||commands[i-1].contentEquals("below") ||commands[i-
1].contentEquals("bottom")   ){destNounY += (Const.objectH/2 + Const.userSize/2);}
                                                //east
                                                if(       commands[i-1].contains("east")
        ){destNounX += (Const.objectW/2 + Const.userSize/2);}
                                                //west
```

```
                                                else if( commands[i-1].contains("west")
        ){destNounX -= (Const.objectW/2 + Const.userSize/2);}

                                                //relational position based off of the user.
                                                //get it:
                                                int quad =
this.getQuad(((User)session.entities.get(0)).cx, ((User)session.entities.get(0)).cy, destNounX,
destNounY, true);



                                                if(commands[i-1].contentEquals("front")
        ||commands[i-1].contains("infront")  ||commands[i].contains("before") ||commands[i-
1].contains("forward")          ||commands[i-1].contains("forwards"))
                                                {
                                                        //if in row column linup
                                                        if(quad == 1){destNounX +=
Const.objectW/2 + Const.userSize/2; destNounY -= Const.objectH/2 + Const.userSize/2;}
                                                        else if(quad == 2){destNounX -=
Const.objectW/2 + Const.userSize/2; destNounY -= Const.objectH/2 + Const.userSize/2;}
                                                        else if(quad == 3){destNounX -=
Const.objectW/2 + Const.userSize/2; destNounY += Const.objectH/2 + Const.userSize/2;}
                                                        else if(quad == 4){destNounX +=
Const.objectW/2 + Const.userSize/2; destNounY += Const.objectH/2 + Const.userSize/2;}
                                                        else if(quad == 5){destNounX +=
Const.objectW/2 + Const.userSize/2;}
                                                        else if(quad == 6){destNounY -=
Const.objectH/2 + Const.userSize/2;}
                                                        else if(quad == 7){destNounX -=
Const.objectW/2 + Const.userSize/2;}
                                                        else if(quad == 8){destNounY +=
Const.objectH/2 + Const.userSize/2;}
                                                }
                                                else if(commands[i-1].contentEquals("back")
        ||commands[i-1].contains("backward")        ||commands[i-1].contains("backwards")
||commands[i].contains("past")        ||commands[i].contains("after")
        ||commands[i].contains("beyond") ||commands[i].contains("behind") )
                                                {
                                                        //if in row column linup
                                                        if(quad == 1){destNounX -=
Const.objectW/2 + Const.userSize/2; destNounY += Const.objectH/2 + Const.userSize/2;}
                                                        else if(quad == 2){destNounX +=
Const.objectW/2 + Const.userSize/2; destNounY += Const.objectH/2 + Const.userSize/2;}
                                                        else if(quad == 3){destNounX +=
Const.objectW/2 + Const.userSize/2; destNounY -= Const.objectH/2 + Const.userSize/2;}
                                                        else if(quad == 4){destNounX -=
Const.objectW/2 + Const.userSize/2; destNounY -= Const.objectH/2 + Const.userSize/2;}
                                                        else if(quad == 5){destNounX -=
Const.objectW/2 + Const.userSize/2;}
```

```
                                                else if(quad == 6){destNounY +=
Const.objectH/2 + Const.userSize/2;}
                                                else if(quad == 7){destNounX +=
Const.objectW/2 + Const.userSize/2;}
                                                else if(quad == 8){destNounY -=
Const.objectH/2 + Const.userSize/2;}
                                        }
                                        else if(commands[i-1].contentEquals("left"))
                                        {
                                                //if in row column linup
                                                if(quad == 1){destNounX +=
Const.objectW/2 + Const.userSize/2; destNounY += Const.objectH/2 + Const.userSize/2;}
                                                else if(quad == 2){destNounX +=
Const.objectW/2 + Const.userSize/2; destNounY -= Const.objectH/2 + Const.userSize/2;}
                                                else if(quad == 3){destNounX -=
Const.objectW/2 + Const.userSize/2; destNounY -= Const.objectH/2 + Const.userSize/2;}
                                                else if(quad == 4){destNounX -=
Const.objectW/2 + Const.userSize/2; destNounY += Const.objectH/2 + Const.userSize/2;}
                                                else if(quad == 5){destNounY +=
Const.objectH/2 + Const.userSize/2;}
                                                else if(quad == 6){destNounX +=
Const.objectW/2 + Const.userSize/2;}
                                                else if(quad == 7){destNounY -=
Const.objectH/2 + Const.userSize/2;}
                                                else if(quad == 8){destNounX -=
Const.objectW/2 + Const.userSize/2;}
                                        }
                                        else if(commands[i-1].contentEquals("right"))
                                        {
                                                //if in row column linup
                                                if(quad == 1){destNounX -=
Const.objectW/2 + Const.userSize/2; destNounY -= Const.objectH/2 + Const.userSize/2;}
                                                else if(quad == 2){destNounX -=
Const.objectW/2 + Const.userSize/2; destNounY += Const.objectH/2 + Const.userSize/2;}
                                                else if(quad == 3){destNounX +=
Const.objectW/2 + Const.userSize/2; destNounY += Const.objectH/2 + Const.userSize/2;}
                                                else if(quad == 4){destNounX +=
Const.objectW/2 + Const.userSize/2; destNounY -= Const.objectH/2 + Const.userSize/2;}
                                                else if(quad == 5){destNounY -=
Const.objectH/2 + Const.userSize/2;}
                                                else if(quad == 6){destNounX -=
Const.objectW/2 + Const.userSize/2;}
                                                else if(quad == 7){destNounY +=
Const.objectH/2 + Const.userSize/2;}
                                                else if(quad == 8){destNounX +=
Const.objectW/2 + Const.userSize/2;}
                                        }
```

```
                                                if(donts > 0){
                                                        this.findMvCnd(cparse, commands,
nouns[j], prevnoun, session.entities.get(0).cx, session.entities.get(0).cy, destNounX, destNounY);

                                                        donts--;
                                                }



        ((User)session.entities.get(0)).addDestination(destNounX, destNounY);

                                                directionOf--;
                                                match = true;
                                                i = prevnoun -1;//out of current noun conditions
detection

                                                if(Const.debug){System.out.println("position
parse");}
                                        }//found an ending position [verb] [end position] [noun]
                                        else if(cparse[i].contentEquals("n dir")        ||
cparse[i].contentEquals("m dir")      ){
                                                //found move in direction until noun
                                                int[] xyMod = new int[2];
                                                boolean nounBlocks = false;
                                                xyMod[0] = xyMod[1] = 0;

                                                //north
                                                if(       commands[i].contentEquals("up")
        ||commands[i].contains("north")
        ||commands[i].contentEquals("above")||commands[i].contentEquals("top")          ){xyMod[1]
= -1; }
                                                //south
                                                else if( commands[i].contentEquals("down")
        ||commands[i].contains("south")
        ||commands[i].contentEquals("below")||commands[i].contentEquals("bottom")    ){xyMod[1]
= 1; }
                                                //east
                                                if(       commands[i].contains("east")
        ){xyMod[0] = 1;}
                                                //west
                                                else if( commands[i].contains("west")
        ){xyMod[0] = -1;}



                                                if(commands[i].contentEquals("front")
        ||commands[i].contains("infront")     ||commands[i].contains("before")
||commands[i].contains("forward")   ||commands[i].contains("forwards"))
```

```
                                                {

        ((User)session.entities.get(0)).getFaceMod(xyMod, 0);
                                                }
                                                else if(commands[i].contentEquals("back")
        ||commands[i].contains("backward")||commands[i].contains("backwards")
||commands[i].contains("past")          ||commands[i].contains("after")
        ||commands[i].contains("beyond"))
                                                {

        ((User)session.entities.get(0)).getFaceMod(xyMod, 2);
                                                }
                                                else if(commands[i].contentEquals("left"))
                                                {

        ((User)session.entities.get(0)).getFaceMod(xyMod, 3);
                                                }
                                                else if(commands[i].contentEquals("right"))
                                                {

        ((User)session.entities.get(0)).getFaceMod(xyMod, 1);
                                                }

                                                int vertMove = 0;
                                                int horiMove = 0;

                                                //if user x is to left of noun x, it needs to go right
                                                //else it needs to go left
                                                if(
        ((User)session.entities.get(0)).cx < destNounX        ){ horiMove = 1;}
                                                else { horiMove = -1;}
                                                //if user y is above noun y, it needs to go down
                                                //else it needs to go up.
                                                if(
        ((User)session.entities.get(0)).cy < destNounY        ){ vertMove = 1;}
                                                else { vertMove = -1;}

                                                //if is not going to be stopped by the noun in either
direction...
                                                //put the destination outside of the map
                                                ((User) session.entities.get(0)).dCount = 0; //first
clear out the check points because position matters.
                                                if ((horiMove != xyMod[0]) && (vertMove !=
xyMod[1]))
                                                {
                                                        destNounX = session.entities.get(0).cx +
(xyMod[0] * 2000);
                                                        destNounY = session.entities.get(0).cy +
```

```
(xyMod[1] * 2000);
                                                        }


                                                        if(donts > 0){

                                                                this.findMvCnd(cparse, commands,
nouns[j], prevnoun, session.entities.get(0).cx, session.entities.get(0).cy, destNounX, destNounY);

                                                                donts--;
                                                        }

                                                        this.setMoveInDirUntil(xyMod[0], xyMod[1],
destNounX, destNounY); //destination matters.. becauase it might not be going in that direction.

                                                        match = true;
                                                        i = prevnoun -1;//out of current noun conditions
detection

                                                        if(Const.debug){System.out.println("direction
parse");}
                                                }//found an ending position [verb] [end position] [noun]
                                        }//end for i > previous noun && >=0

                                        if(!match){
                                                parseInfo = "Improper structure";
                                        }

                                        //immediately update prevnoun.
                                        prevnoun = nouns[j];


                                }//end for j < nounpos


                        }//there was a noun
                        else{
                                parseInfo = "No nouns detected";
                        }
                }
                else{
                        parseInfo = "No verbs detects. ";
                }

                for(int i = 0; i < cparse.length; i++){
                        if(cparse[i].length() > 0) {parseInfo = parseInfo + "[" + cparse[i] + "]";}
                }
                parseInfo = parseInfo + " Press <Enter> to continue.";
```

```
	}

	/**
	 * Finds the position of the "mv cnd"
	 *
	 *
	 * @return
	 */
	public int findMvCnd(String[] parse, String[] orig, int start, int end, int ux, int uy, int dx, int
dy){
		int retval = -1;
		int nval = -1;
		boolean foundMatch = false;
		int nounX = 0;
		int nounY = 0;

		//first search to the left
		for(int i = start; i > end; i--){
			if(parse[i].contentEquals("mv cnd")){retval = i; i = end;}
		}
		//if wasn't found
		if(retval == -1){
			//go right
			for(int i = start + 1; i < parse.length; i++){
				if(parse[i].contentEquals("mv cnd")){retval = i; i = parse.length;}
				else if(parse[i].contentEquals("verb") || parse[i].contains(" n")){retval =
i; i = parse.length;}
			}
		}
		if(Const.debug){System.out.println("Inside mv cnd find\nRetval: " + retval);}
		//if mv cnd was found
		if(retval != -1){
			//need to find the associated verb
			for(int i = retval + 1; i < parse.length; i++){
				if(parse[i].contentEquals("mv cnd loc")){nval = i;}
			}

			if(Const.debug){System.out.println("nval: " + nval);}
			if(nval != -1){
				foundMatch = true;
				if(this.isType(treeList, orig[nval])){nounX =
session.entities.get(Const.aTree).cx; nounY = session.entities.get(Const.aTree).cy;}
				else if(this.isType(fountainList, orig[nval])){nounX =
session.entities.get(Const.aFountain).cx; nounY = session.entities.get(Const.aFountain).cy;}
				else if(this.isType(catList, orig[nval])){nounX =
session.entities.get(Const.aCat).cx; nounY = session.entities.get(Const.aCat).cy;}
				else if(this.isType(lightList, orig[nval])){nounX =
session.entities.get(Const.aLight).cx; nounY = session.entities.get(Const.aLight).cy;}
```

```
                                else if(this.isType(buildingList, orig[nval])){nounX =
session.entities.get(Const.aBuilding).cx; nounY = session.entities.get(Const.aBuilding).cy;}
                                //if(this.isType(treeList, orig[nval])){nounX =
session.entities.get(Const.aTree).cx; nounX = session.entities.get(Const.aTree).cy;}
                                else{
                                        foundMatch = false;
                                }

                        }

                        //if there was a match
                        if(foundMatch){
                                //create rectangles and lines representing path and obstacle
                                Rectangle obstacle = new Rectangle(nounX - Const.objectW/2, nounY -
Const.objectH/2, Const.objectW/2, Const.objectH/2);
                                Line2D.Double leftLine = new Line2D.Double(ux - Const.userSize/2,
uy, dx - Const.userSize/2, dy );
                                Line2D.Double midLine = new Line2D.Double(ux, uy, dx, dy );
                                Line2D.Double rightLine = new Line2D.Double(ux + Const.userSize/2,
uy, dx + Const.userSize/2, dy );

                                boolean lhit, mhit, rhit;
                                //see if they intersect(meaning collision)
                                if(leftLine.intersects(obstacle)){lhit = true;}
                                else{lhit = false;}
                                if(midLine.intersects(obstacle)){mhit = true;}
                                else{mhit = false;}
                                if(rightLine.intersects(obstacle)){rhit = true;}
                                else{rhit = false;}

                                //if there was a collision
                                if(lhit || mhit || rhit){
                                        //first figure out where the user is in relation to the obstacle
                                        int quad = this.getQuad(ux, uy, dx, dy, false);

                                        //if quad == 1
                                        if(quad == 1){
                                                //both hit. Go around either way
                                                if(!lhit && rhit)
                                                {

        ((User)session.entities.get(0)).addDestination(nounX - Const.objectW/2 - Const.userSize/2,
nounY - Const.objectH/2 - Const.userSize/2);

        ((User)session.entities.get(0)).addDestination(nounX - Const.objectW/2 - Const.userSize/2,
nounY + Const.objectH/2 + Const.userSize/2);
                                                }
                                                else
```

```
                                        {

        ((User)session.entities.get(0)).addDestination(nounX + Const.objectW/2 + Const.userSize/2,
nounY + Const.objectH/2 + Const.userSize/2);

        ((User)session.entities.get(0)).addDestination(nounX - Const.objectW/2 - Const.userSize/2,
nounY + Const.objectH/2 + Const.userSize/2);
                                        }
                                }
                                if(quad == 2){
                                        //both hit. Go around either way
                                        if(!lhit && rhit)
                                        {

        ((User)session.entities.get(0)).addDestination(nounX + Const.objectW/2 + Const.userSize/2,
nounY - Const.objectH/2 - Const.userSize/2);

        ((User)session.entities.get(0)).addDestination(nounX + Const.objectW/2 + Const.userSize/2,
nounY + Const.objectH/2 + Const.userSize/2);
                                        }
                                        else
                                        {

        ((User)session.entities.get(0)).addDestination(nounX - Const.objectW/2 - Const.userSize/2,
nounY + Const.objectH/2 + Const.userSize/2);

        ((User)session.entities.get(0)).addDestination(nounX + Const.objectW/2 + Const.userSize/2,
nounY + Const.objectH/2 + Const.userSize/2);
                                        }
                                }
                                if(quad == 3){
                                        //both hit. Go around either way
                                        if(!lhit && rhit)
                                        {

        ((User)session.entities.get(0)).addDestination(nounX - Const.objectW/2 - Const.userSize/2,
nounY - Const.objectH/2 - Const.userSize/2);

        ((User)session.entities.get(0)).addDestination(nounX + Const.objectW/2 + Const.userSize/2,
nounY - Const.objectH/2 - Const.userSize/2);
                                        }
                                        else
                                        {

        ((User)session.entities.get(0)).addDestination(nounX + Const.objectW/2 + Const.userSize/2,
nounY + Const.objectH/2 + Const.userSize/2);

        ((User)session.entities.get(0)).addDestination(nounX + Const.objectW/2 + Const.userSize/2,
```

```
nounY - Const.objectH/2 - Const.userSize/2);
                                                }
                                        }
                                        if(quad == 4){
                                                //both hit. Go around either way
                                                if(!lhit && rhit)
                                                {

        ((User)session.entities.get(0)).addDestination(nounX - Const.objectW/2 - Const.userSize/2,
nounY + Const.objectH/2 + Const.userSize/2);

        ((User)session.entities.get(0)).addDestination(nounX - Const.objectW/2 - Const.userSize/2,
nounY - Const.objectH/2 - Const.userSize/2);
                                                }
                                                else
                                                {

        ((User)session.entities.get(0)).addDestination(nounX + Const.objectW/2 + Const.userSize/2,
nounY - Const.objectH/2 - Const.userSize/2);

        ((User)session.entities.get(0)).addDestination(nounX - Const.objectW/2 - Const.userSize/2,
nounY - Const.objectH/2 - Const.userSize/2);
                                                }
                                        }


                                }//end if there was a collision

                        }//end if found match

                }//end if mv cnd was found


                if(!foundMatch && retval!= -1){parseInfo += "*Unpaired [mv cnd]*";}



                return retval;
        }//end findMvCnd


        /**
         *
         *Four parameters for two sets of x,y.
         *based upon positioning return quad.
         *edit: added 4 additional quads to represent column/row linup.
         *
         * @param x1 reference object
```

```
 * @param y1
 * @param x2 origin object
 * @param y2
 * @return
 */

public int getQuad(int x1, int y1, int x2, int y2, boolean zone8){
	int quad = 0;
	char overlap;
	if(zone8){overlap = isColRowOverlap(x1, y1, x2, y2);}
	else{overlap = ' ';}

	//if user x is greater than object x. user is right of object
	if(x1 > x2){
		if(overlap == 'x'){quad = 5;} //if right of and overlap
		else if(y1 > y2){
			if(overlap == 'y'){quad = 8;} //if right of and below
			else{quad = 4;}
		} //if right of and below
		else {
			if(overlap == 'y'){quad = 6;} //if right of and below
			else{quad = 1;}
		} //if right of and below
	}
	//else if x1 is to left of x2
	else {
		if(overlap == 'x'){quad = 7;} //if right of and overlap
		else if(y1 > y2){
			if(overlap == 'y'){quad = 8;} //if right of and below
			else{quad = 3;}
		} //if right of and below
		else {
			if(overlap == 'y'){quad = 6;} //if right of and below
			else{quad = 2;}
		} //if right of and below
	}


	return quad;
}//end get quad

/**
 *Four parameters for two sets of x,y.
 *will check if column or row position overlaps
 *set 1 assumed to be user, set 2 assumed to be object.
 *
 *@ return 'x' 'y' and 'b'(both)
 *
```

```
*/
public char isColRowOverlap(int x1, int y1, int x2, int y2){
	char overlap = ' ';

	int dx = Math.abs(x1 - x2);
	int dy = Math.abs(y1 - y2);

	if(dx < Const.userSize/2 + Const.objectW/2){ overlap = 'x'; }
	if(dy < Const.userSize/2 + Const.objectH/2){
		if(overlap == 'x'){overlap = 'b';}
		else{overlap = 'y';}
	}

	return overlap;
}//end get quad

//used to detect if a word is in a list
//given list and word to check against
public boolean isType(String[] list, String word){
	boolean retval = false;

	for(int i = 0; i < list.length; i++){
		if(word.contentEquals(list[i])){
			retval = true;
			break;
		}
	}
	return retval;
}//end isType

//detects commands.
//current format:
//count nouns. Will need further exclusion based off of conditional moves.
public String[] detectCommands(){
	String[] retval;
	String[] words = userInput.split(" ");
	int[] commandPoints = new int[1];
	int verbs = 0;

	for(int i = 0; i < words.length; i++){
		for(int j = 0; j < verbList.length; j++){
			if(words[i].contentEquals(verbList[j])){
				verbs++;
				if(verbs > commandPoints.length){
					int[] tempA = new int[commandPoints.length * 2];
					for(int k = 0; k < commandPoints.length; k++){
						tempA[k] = commandPoints[k];
					}
```

```
                                        commandPoints = tempA;
                                    }//end if verbs
                                }//end w
                            }//end j
                    }//end i


                    if(commandPoints.length > 1){

                    }
                    //else{
                    retval = new String[1];
                    retval[1] = userInput;

                    return retval;
            }//end detect commands.

            //counts the number of verbs in a sentence.
            public int countVerbs(String sentence){
                    int retval = 0;

                    return retval;
            }

            //counts the number of nouns in a sentence.
            public int countNouns(String sentence){
                    int retval = 0;

                    return retval;
            }


            //calculate direction move until parameter given or screen boundaries reached
            //natural limit at screen boundaries.
            public void setMoveInDirUntil(int xmod, int ymod, int xdest, int ydest){
                    int ux, uy;
                    boolean spotMatch = false;

                    if(xmod == 0 && ymod == 0){
                            if(Const.debug){
                                    System.out.println("Move cancelled due to mods being 0");
                            }
                            return; //can't move if there is no mod value.
                    }

                    ux = ((User)session.entities.get(0)).cx;
                    uy = ((User)session.entities.get(0)).cy;
```

```
                while(!spotMatch){
                        if(xdest * xmod == ux || ydest * ymod == uy){
                                spotMatch = true;
                        }
                        else if(Const.wwidth < ux + Const.userSize/2){
                                ux = Const.wwidth - 1 - Const.userSize/2;
                                spotMatch =true;
                        }
                        else if(0 > ux - Const.userSize/2){
                                ux = 1 + Const.userSize/2;
                                spotMatch = true;
                        }
                        else if(Const.wheight < uy + Const.userSize/2){
                                uy = Const.wheight - 1 - Const.userSize/2;
                                spotMatch = true;
                        }
                        else if(0 > uy - Const.userSize/2){
                                uy = 1 + Const.userSize/2;
                                spotMatch = true;
                        }
                        else{
                                ux += xmod;
                                uy += ymod;
                        }
                }//end while no spot match


                ((User)session.entities.get(0)).addDestination(ux, uy);

        }//end set move in dir until

        //get string methods
        public String getIM(){ return inputMarker;}
        public String getPI(){ return parseInfo;}
        public String getUI(){ return userInput;}

        //get text start area
        public int getTextAHS(){return workAreaH;}
        //get state
        public boolean isPolling(){return gettingInput;}


}//end class
```