City Infrastructure Advancement Proposal

Draft ID: CH024

2008

Contents

Chapter 1

Introduction

Chapter 2

On the roads

2.1 Car physics

In our modern neoprimitivist times, cars look pretty simple. They have a shape of a rectangle. The length of a car is 3.2 meters, the width is 2 meters. Surprisingly enough, cars have two wheels. One is located at the middle of the front of the car, the other is at the middle at the back of the car. Despite this deficiency, cars never tilt as motorcycles do, thanks to some engineering magic. This magic does not affect the movement of the cars any other way. The cars are moved by forces acting between its wheels and the ground. You can affect these forces four different ways: you can turn the first wheel between -45°and 45°. You can set the throttle which controls the power of the engine. You can also set the gear, which controls the connection between the engine and the wheels. And finally you can press the brake with some force.

2.1.1 For theoretists

You can experiment to find out how cars work, but if you are rather a theoretical type, here are some basic principles. All forces moving the cars are transfered through the wheels, so the easiest thing is to outline the forces that are acting on the them.

If the throttle is not zero, then the engine tries to move the wheels forward. The strength of the engine depends on the actual speed of the wheels (not the speed of the car, that can be different from the speed of the wheel if the tires have slipped!). The strength of the engine is maximal if the wheels are stopped or turning to the opposite direction of the engine's force. The faster the wheels turn forward, the weaker the engine gets, using a linear function. (If the gear is in rear mode, then the faster the wheels turn backward the weaker the engine gets.) The actual force also depends on the transmission. If you switch to higher gear, the initial force gets lower (proportionally to the gear ration) but the slope of the force decreasing gets lower (proportionally to the square of the gear ratio). So in higher gear you have less force in slow motion, but the maximal speed gets higher.

If the brake force is set to non-zero value, then the brake tries to stop the wheels, that is acting opposite to their actual turning direction, or trying to hold it still if they are not moving (relative to the car, not to the road).

Finally the road also acts on them. As long as the tires do not slip, there is a constraint force that moves the wheels so that their surface remains fixed relative to the ground. Once the wheels have slipped, they remain sliding as long as the speed of their surface is moving relative to the ground. In this case a constant size force acts on the wheel surface, pulling the wheel to the opposite of the velocity of the wheel's surface. The strength of this friction force is very small (only a fraction :)) compared to the maximal constraint force that can act in the first case. That's the main reason why you loose control when your wheels slip, so you want to avoid it mostly. The forces of the ground acting on the two wheels just mentioned above are the forces that are finally moving your car.

The wheels get slipped if the constraint force to keep them still would become two large. The force falling on a wheel depends on many factors, you can make your wheels slip for example by turning too fast, turning the steering wheel too quickly, braking or accelerating too fast, or any combination of these.

The two wheels are separate, can be imagined to have separate engines, and also it is possible that only one wheel slips.

2.2 Map syntax and semantics

The simulation server awaits it's customers on four ports. One is for setting up and controlling a city, the second is

2.2.1 Elements of a city

The area of a city – a rectangle in these modern times – is partitioned into equal size square cells. Each cell is 4 meters by 4 meters large. Each square is covered by one of the city elements. We will use coordinates in many cases to refer to specific points of the city (not to specific cells!). The coordinate (x, y) will refer to the point that is x meters away from the western edge of the city and y meters away from the southern edge. When we refer to angles, we always give them in radians, and you also have to use these coordinates and angles when communication with the simulation server.

In our motorized society the most important cell type is the road segment. A road segment cell is a part of a track of a road, that is only one way traffic is allowed on it. But it is possible that cars should turn on a segment. Normal two way roads are established by putting two rows (columns) of road segments next to each other, with opposite allowed directions. Each segment has two parameters: the side in which entering the cell is allowed and the side in which cars have to exit the cell. For example, if a road segment's entering direction is south and exiting direction is west, then cars are supposed to turn left on this cell.

To make cars' jobs harder, a city also has intersections. There is a special cell type corresponding to the ground of an intersection. Typically more than one such builds up the area of an intersection. There is one more special element, that is a road segment entering an intersection, that is a road segment from where cars reach the area of the intersection. These segments are straight, that is they are either south-north, north-south, east-west or west-east oriented. They also specify the set of possible routes a car can take in the intersection, it is a subset of the possibilities right, straight and left. There are four different types of them. These are "give way to the right", "traffic light", "yield" and "priority".

The last type of road cell is that of the fork. It is used when one lane splits into two. It has one entrance and two exit directions.

Houses are also important elements of cities – they are mainly used to crash cars on their walls. There are many sorts of houses, marked with numbers from 0 to 9. Houses of sort 0 are the smallest, sort 9 houses are the tallest.

Finally there are parking places. The only place where cars can legally be stopped, except for traffic reasons.

It is sometimes necessary to represent incomplete maps. For these we use empty cells.

2.2.2 Syntax of the map

To describe the map of a city, we use the following format. Each line corresponds to a row of cells of the city, listed from west to east. Each cell description consists of a cell type id and a parameters string. The cell type id is one of the numbers 0-7 (see below). The parameters string is explained type by type below. The two parts and the different cells are separated by a semicolon (;). The encoding of the different types goes like this:

Road segment Type 0. Parameter string is two characters long, each character is one of n, s, e, w, specifying the entering and exiting directions respectively.

Intersection area Type 1. Parameter string is empty.

Intersection entrance with traffic light Type 2. Parameter string consists of three parts separated by colon. The first is an identifier string of the light. The second is one of the strings ns, sn, ew, we specifying the direction of the segment. The third is a specification of the possible outgoing directions from the intersection if we entered using this cell. It is one or more of the characters r, s, l.

Intersection entrance with give way to right rule Type 3. Has two parameters separated by colon, the first is the direction, the second is the outgoing options specified the same way as the second and third parameters above.

Intersection entrance with yield rule Type 4. Parameters as above.

Intersection entrance with priority rule Type 5. Parameters as above.

Fork Type 6. Three parameters separated by colons. First is the entrance direction, second and third are the two exit directions.

House Type 7. One of the characters 0-9, specifying the sort of the house.

Parking 1 Type 8. Parameter string is empty.

Parking 2 Type 9. Parameter string is empty.

Empty Type 10. Parameter string is empty.

2.2.3 Tidy maps

Any two-year-old can design a city, one would think. But probably the city will look like a complete mess. We will call a city map tidy, if it follows a few rules:

(1) There are no cells of type 10.

(2) The cell at (one of) the exit direction(s) of a cell type 0 or 6 should be one of types 0, 2, 3, 4, 5 or 6 (specially still be on the map) and the entrance direction of this neighboring cell is the opposite of the original cell's above mentioned exit direction.

(3) The cell at the entrance direction of a cell type 0 or 6 should be one of types 0, 1 or 6 (specially still be on the map) and (one of) the exit direction(s) of this neighboring cell is the opposite of the original cell's entrance direction.

(4) At the entrance direction of a cell type 2, 3, 4 or 5 there must be a cell type 0 with the appropriate exit direction.

(5) At the exit direction of a cell type 2, 3, 4 or 5 there must be a cell type 1.

(6) Road segments must be aligned following the rule that cars go on the right side of the road. For example, if we have a sn, sw or wn cell of type 0, 2, 3, 4 or 5 or a snw cell of type 6, then the cell to the east must be one of sn, en, se of type 0, 2, 3, 4 or 5 or a sne cell of type 6 or a type 7 cell. All the rotated versions of this rule apply.

(7) If there is any cell north of a sew type 6 cell, then it must be of type 7.

(8) All intersections (areas covered by type 1 cells) must be rectangular.

(9) At each side of an intersection rectangle there must be at most one type 0 cell, all others should be of type 2, 3, 4, 5 or 7. We cannot mix type 2, 3, 4 and 5 cells on the same side of the rectangle.

(10) The road type cells (types 0, 2, 3, 4, 5) must reside continuously at any side of the rectangle (that is type 7 cells cannot reside between road type cells).

(11) Each turning direction (r, s, l) must appear in at most one of the type 2, 3, 4, 5 cells in one side of the rectangle, that is you can go into each direction from at most one lane.

(12) The order of the assignment of the r, s and l directions must be in a logical order: no r allowed to left of an s or l, and no s allowed left of an l. But even all characters can appear on the same cell, meaning only one entrance to the crossing.

(13) If an intersection is neighboring to a type 2 cell, then it has no neighbor of type 3, 4 or 5.

(14) If an intersection is neighboring to a type 3 cell, then it has no neighbor of type 2, 4 or 5.

(15) If there is a type 4 cell on a side of the rectangle, then the opposite side can not contain cells 2, 3 and 5, and the sides to the left and right must contain type 5 cells.

(16) If there is a type 5 cell on a side of the rectangle, then the opposite side must contain at least one cell of type 5, and the sides to the left and right must not contain type 2, 3 and 5 cells.

(17) The entrance cell containing the s character must be in front of the only type 0 cell on the opposite side (meaning also that there must be a type 0 cell on the opposite side)

(18) If there is an r (l) character on one side, then there must be a type 0 cell on the side to the right (left).

(19) All intersections must have road segments on at least three of its sides.

(20) Each type 8 cell must be a neighbor of one straight type 0 cell. Staying at this type 0 cell, looking at the direction of the type 0 cell, the parking cell must be to the right. Moreover, let's assume that the above mentioned type 0 cell is north from a type 8 cell. Then east and west of the type 0 cell, there also must be two straight type 0 cells. The rotated versions apply.

(21) Each type 9 cell must be a neighbor of a type 8 cell. It must be on the opposite side of the type 8 cell as the above mentioned type 0 cell.

(22) Other than specified by the above two rules, type 8 and 9 cells can have only houses and other parking spaces as neighbors.

2.3 Traffic code

The traffic in the city is regulated by very strict rules to ensure the safety of the (robot) drivers. Now a quite formal description follows. But it is basically the normal traffic code, with some differences. A vehicle can be in one of the following modes: normal moving, parking, starting.

The rules of normal moving:

(1) The car must be moving on the map using only cells 0, 1, 2, 3, 4, 5 and 6.

(2) Any point of the car's body must enter a cell only on the cell's entrance side and exit on one of it's exit sides. All points must use the same exit side to exit.

(3) When going through an intersection, no point of the car's body can cross the the following lines. One we get by extending the left side of the arrival lane. The second we get by extending the left side of the exiting lane. (Notice that these two are the same by the rules of tidy maps in case of a straight crossing through the intersection.) If the car goes straight, it also cannot cross the line which we get by extending the right side of the arrival lane (which is the same as the right side of the exiting lane.) We call the area of the intersection which the car is allowed to touch the allowed area for that car.

(4) Normally the car's speed must always be between !!! and !!! km/h

(5) The only exception from the above rule is when the way of the car is blocked by an other car, see the next few rules.

(6) If car1 is moving in front of car2 to the same direction, then car2 has to maintain such a distance that in case car1 would start braking with full strength, then car2 can still react and brake fast enough to never get closer then half a square to car1.

(7) If there are at least two empty cells in the car's planned way (not counting those occupied by the car itself), then the car has to already be going with !!! km/h. (That is, it is too late to start accelerating when two cells become empty in front of the car.)

(8) After leaving an intersection or fork, you have to signal your direction at the next intersection or fork with your index.

(9) Entering an intersection is the moment when the first point of the car moves above a type 1 cell.

(10) Cars should handle cars blocking their way in intersections as on normals roads if not otherwise specified. That is, they have to keep the distance from cars going in front of them, but they don't need to count on cars entering their way from the sides, as long as they keep the following entrance rules.

(11) When approaching an intersection through a traffic light, the car is allowed to enter the crossing if either:

The light is green

The light is orange and it was green at the moment when the car entered the intersection's entrance cell

(12) When approaching an intersection through a type 5 entrance, you can always enter if you go straight or right

(13) In all other cases, there might be cars for which you have to yield (see below). You can enter an intersection if you are sure to be able to leave it before a car that has the right of way and whose allowed area intersects with yours can enter it. You can assume that all car go with at most !!! km/h.

(14) In case the above rule would cause a dead lock, that is all cars waiting for come other, then the car first arrived can go. If there is a draw, then among the drawers the car arriving from south has the priority, then the order is east, north, west. You can enter using this rule only if you can exit, that is the exit way is not blocked.

(15) In type 3 entrance, cars coming from right and front has the right of way over you (remember, it only matters if your allowed areas actually intersect!)

(16) In a type 4 entrance, cars coming from any direction has the right of way over you.

(17) Turning left from a type 5 entrance, cars coming from front has the right of way over you.

Rules of parking:

(1) When you park, your car is located on the area occupied by two cells: one of type 8 and one of type 9.

(2) When you park, your car is stopped.

(3) When approaching a parking place, you can only use road segments normally and the two parking areas you are going to use. You can slow down below the minimal allowed speed at most two squares before the parking slot.

(4) You can only use completely empty parking cells for parking.

(5) You have to set your index to right after leaving the last intersection/fork before parking

Rules of starting:

(1) You start from a parking slot by first going back and turning left onto the lane neighboring and perpendicular to the parking slot, and then start going straight on the lane, resuming normal speed.

(2) You cannot touch any other road lane then the one closest to your spot.

(3) You can start from a parking slot if you can maneuver yourself to normal moving with normal speed before any car gets closer to you then two squares.

(4) From the moment when any point of your car touches the road lane, your index has to be set according to your direction at the next intersection or fork.

2.4 The simulation server and its services

The simulation server welcomes its customers on four different ports, giving different services. The master port is for creating new cities and controlling them. The car controlling port is for attaching to existing cars in existing cities and then control them. The observer port is for following the status of a given city. And finally light control port is to change the state of the traffic lights.

2.4.1 The master port

When you connect to the port 8007 of the server, you will get a welcome message, asking for the name of the city you want to create. If the name is already taken, then you are asked again.

After successful selection of a city name, you will have to send the map of the city as specified in section 2.2.2. An empty line after the map rows signals that the map is over.

After getting through these two steps, your city is set up, and you can issue the following commands:

start Starts the simulation of the city. Everything gets moving!

stop Stop the simulation. Save some CPU time for other teams...

step num Make the simulation step *num* steps.

notify Send the current status of the city to all observers.

createcar name x y angle Creates a car called *name* in the (x, y) point facing to the direction *angle* from east.

exit Exits the simulation. All observers and controllers affiliated with this city will be disconnected. Aware the city will be completely destroyed, that is its state is lost. When the master disconnects it has the same result.

2.4.2 The observer port

When you connect to the port 8008 of the server, you will get a welcome message, asking for the name of the city you want to observe. If the name is not known as an existing city, then you are asked again. If you manage to enter the name of an existing city, then you have no more say in this protocol. But the server will keep you informed of all the relevant events of the city. You will have these kind of messages:

LIGHT name SET TO value The light identified by *name* was switched to state specified by *value*. 0 is green, 1 is orange, 2 is red.

TIME time The server's timestamp is *time*. It will be sent before each complete status description. One server tick corresponds to $\frac{1}{500}$ second.

BEGIN A complete status description will arrive. Lines describing the status of each car and each light will follow.

END Marks the end of a complete status description

CAR name:x:y:a:vx:vx:w The car called *name* is at coordinates (x, y) (the center of the car, in meters), facing at a radians from east, moving with (vx, vy) velocity (in meters/second), and rotating with w angular velocity (in 1/second).

LIGHT name:value The light called *name* is in status specified by *value*.

You have to be prepared to encounter different kind of messages. These should be simply ignored. This is to allow for further enhancements to the server.

2.4.3 The car controller port

When you connect to the port 8009 of the server, you will get a welcome message, asking for the name of the city and the name of the car you want to control. If the city name is not known as an existing city, or the car name is not valid then you are asked again. If you manage to enter the name of an existing city and existing car, then you can start giving commands to your car. These are the available commands:

throttle x Sets the throttle to x. Must be in $[0, 1]$.

brake x Sets the brake to x. Must be in $[0, 1]$.

gear x Sets the gear to x. Must be in $\{-1, 1, 2, 3\}$.

index x Sets the index to x. Must be in $\{-1, 0, 1\}$. 1 means left, 0 forward, -1 right.

steer x Sets the first wheel to angle x (in radians). Must be in $[-\pi/4, \pi/4]$.

Chapter 3

Where can you help?

3.1 Municipal contract system

Building a metropolis is a very complex and challenging task, and needs a good cooperation of many individuals and companies. Here is the system invented by the mayor to keep the construction running smoothly.

3.1.1 Getting approved

Some of the tasks are distributed using a tender system. At the beginning of the construction (codename Challenge 24), the specification of various projects are distributed among all potential contractors (aka teams). The projects are also assigned a priority number. After a consideration period of one hour, the projects are assigned one by one starting with the higher priority projects. In each round, each team makes a bid for the time frame they are able to complete the project (or they can also pass a project). After all the bids are secretly collected the city hall reviews them and announces the first at least 10 teams to be appointed to the project. After this, the same procedure is repeated for the next project. Every team is allowed to implement any project, and they are going to still get the base points for those that are successfully developed. But the appointed contractors have some possibility to earn some extra points – and also to loose some. If the project is completed in the time frame they have agreed to finish it they get a bonus of 30% of the base points. If they submit the project late, their bonus reduces by 10% every half hours, also going to negative in case of a large late. For example, if you submit your project 2 hours late, you will get 90% of the base score.

As always, you will not need to create programs that the organizers can actually run. But still there will be a submission server. When you finish a project, you have to zip together all the relevant source files and submit it to the server. When your project is judged, you will need to convince the judge that you are showing the submitted version (basically download it from the server, and compile/run the downloaded files). You are always free to resubmit your project to the server and decide which version to show to the jury – of course if you choose a later version, if any bonuses/penalties apply, they will be computed using the time of the later submission.

If your project gets rejected because of some error, you will have a penalty of 30% of the base score (it is in addition to any bonus earned from the contractor system). If the project gets scored using some subpoints, you might get the penalty only for the affected subscores. After the failed submission attempt, you will have an extra hour to resubmit your fixed project, and then a new submission takes place. If you does not submit in an hour from the rejection, it is counted as a new failed submission, and the clock automatically restarts at the end of the hour.

Even if you are an assigned contractor, you will never get a negative score for a task.

3.1.2 Cross-validation

In some cases teams are asked to check other teams' solutions with some checking programs. In this case both the projects and the checking programs are being examined. You will need to put together your solution with the other team's checker, and then come to an agreement on whether the programs are correct or not. As there will be clear specifications, there should be no problem about making this decision, but in case of any debate ask the organizers. After this checking session both teams have to upload the results – that is whether the program and whether the checker is correct – on the web interface. Of course the results uploaded should be the same. In case of a failure accepted by the author team, the resubmission deadline is one hour from the validation session. Please as this is a question of fair play, signal your failure right after the test – and also signal the opposing teams failure quickly.

3.2 Building new cities

The principal architect of the city needs your help badly! It is not very easy to design a tidy city map fulfilling all the requests of the various lobby groups. So your task is to create tools to help him in his creative and challenging task. He will be very thankful...

3.2.1 What he sees is what he gets

Your first task is to create a graphical map editor. The editor should be able to save and load maps, start an empty map and be able to create any possible map with just a few easy mouse clicks. A new map should contain a special "empty" cell at all positions, which can then be changed one by one or in groups to the various possible map elements described in 2.2. The user has to be able to specify all properties explained and you need to visualize these to allow a quick overview.

It is NOT part of this task to check the map for being tidy. In fact it should allow for all possible ugly and untidy maps, too. But, on the other hand it should be able to consult a network checker. The user specifies a host and the port, then the program connects to the specified port, and sends the current status of the map in the editor in the format specified in section 2.2.2. The host can be expected to send back a text report and then close the connection. Most of this text just has to be displayed for the user without any modification. But there can be some number of character sequences which has to be treated specially. They look like `#ref(x,y,type)#` where `x,y` is a pair of integers in the coordinate space of the map and type is an integer in the range 0-9. If we encounter a sequence like that, we have to replace it in the text with `x,y`, and highlight the cell x,y on the map somehow. All types should have somewhat different highlight. The user should have an option to delete the highlights afterwards.

3.2.2 Checking a map for tidiness

Your task is to create a network server program which is able to check a city map for tidiness. Your service should accept connections on port !PORT!, and follow the protocol explained in the previous section. You need to generate a human readable report pointing out all violations of rules in section 2.2.3. You also need to put `#ref(x,y,type)#` sequences to highlight the violating coordinates, using different types for different rule violations.

3.2.3 Completing the plan

Sometimes the principal architect just gets bored of the city at hand. You are to create a tool which he can use in these cases. You have to create a service which listens at port !PORT!. First it gets a map described using the format specified in section 2.2.2. The uploaded map might have some empty cells. The program returns a map where no cells are empty and is tidy, or returns the string `Don't ask the impossible!` if no such completion exists. In case there are more possible completions, the program should be controllably random as explained in the 3.1.

3.2.4 Judging

3.3 Surveillance systems

The mayor of the city has some kind of strange affection to the idea of being the Big Brother of everyone in his city. So he wants you to implement a complete surveillance system of the streets of the city.

3.3.1 2D surveillance

Create a program that connects to the simulation server as an observer and shows the whole or part of the city map with all the cars moving along the streets. As cities tend to be large, an option to show only part of the city at a time is a must. But we also would like to have an overview of the whole city. In the previous mode, you have to enable the user to scroll manually, or to select a car which the map follows automatically. You also have to enable to mark up to ten cars with different skins, to be able to distinguish them easily. The user can select these cars by clicking on them or by selecting from a list of all car names (both options should be implemented). You should display different kind of buildings differently and make connected and unconnected buildings look different.

3.3.2 3D surveillance

Create a 3D visualization program that connects to the simulation server and displays the city from a given car's driver's point of view. You can assume the height of the cars as you wish, but they should be lower then any building. You should display different kind of buildings differently and make connected and unconnected buildings look different. You can display extra, randomly generated items on your view, like pedestrians, people behind windows, graffities or whatever you can imagine for extra points. This task can be done integrated with the previous one. That is, if you can display the city in 3D and allow the camera to be put above the city and have an overview as asked for in the previous section then you can get the points for both tasks.

3.3.3 Judging

3.4 Driving nicely

The safety of the city is of utmost importance. So we need tools to enforce the traffic code. But even if we have those, we cannot leave such a critical task as driving to so untrustable agents as human beings... So you will also write a program that connects to the simulation server and drives a car according to all the rules outlined in section 2.3.

3.4.1 Traffic code checker

Write a program that connects to the simulation server as on observer. It follows the movement of every car and displays timestamped messages of all the violations that the cars do on the streets.

3.4.2 Robot pilots

The robot pilots should have two modes. In one mode they are connecting to a navigator service. The protocol of this service is as follows. In each turn the robot pilot sends its current grid coordinates to the server as two integers separated by a space character (the first is the west-east ordinal of the cell, starts with 0 at the westmost cells, the second is the south-north ordinal, starts with 0 at the southmost cell.). The position should be either a road segment on which the car is moving in the right direction or a parking place where the car is currently parking. The navigator responds with at most three characters without any delimiter characters in a separate line. Each of the characters is one of `r`, `l` or `f` which specifies the direction the car should take at the following next three crossings. In case when the starting position is a parking place, the first character specifies which direction the car should start on the street, and can only be either `r` or `l`. You can trust that the navigator will not specify a direction that is not available at a given crossing. You should not call the navigator more than once without going through a crossing. It would not make any sense anyways. You can count on the navigator not to change his mind about the already given directions.

The second mode is to select each direction using controlled random choices and crawl the city endlessly.

The robot pilots should be able to use a resumer if one's address is specified on startup (see section 3.4.4).

3.4.3 Navigator

Your task is to implement a navigator as specified in the previous task. The input of the navigator specified on startup should be the position of one parking spot. Then the navigator should navigate any car connecting to its port to the given parking slot. The navigator should be able to recalculate (one of) the best path from any requested position, it should not assume anything about the requests (such as the next request may not be located somewhere on the previously advised path).

There are two category of navigators. The first should simply compute and advise the shortest path from the requested starting point to the end spot. The length is measured in the number of cells touched on the route.

The second category also has to act as an observer. It has to navigate also taking traffic considerations into account. These navigators will be judged by navigating contests.

3.4.4 Resuming

A resumer is a special car controller. It is used after a car got involved in an accident. If this happens, the primary controller should execute the following sequence. First set the brakes to maximum force, and wait until the car stops moving. Then, if available, a resumer should be invoked. This is done by first disconnecting the primary controller from the simulation server (releasing the control of the car), then connecting to the resumer, and sending it a single line, the name of the city and the car separated by a space character. At this point the resumer has to gain control over the car, and drive it to a normal driving position. Typically stopped on a road segment, facing to the right direction. At this point the resumer disconnects from the simulation server and writes the line "DONE" back to its invoker. Then the primary controller takes over again, and continues as before the accident.

The resumer has to be created in a way that it does not hit any car which takes part in the traffic obeying the traffic code. It might be very hard to write a perfect resumer, so the scoring is going to be done by a resuming contest.

3.5 The secret racing league

Although safe driving is safe, it is also pretty boring. Why don't you participate in our racing league, where there are no rules whatsoever! There will be three different championships during the contest at hours 8, 16 and 24, drivers will gradually face harder and harder tasks.

3.5.1 Empty streets

At first you have to drive between two given points in an empty city as fast as possible. The competition proceeds as follows. You have to listen to a special competition port. When your turn comes, the name of a city and a car and the grid coordinates of a road cell will be sent to you. This time you can connect your controller and observer. The simulation will start 30 seconds after the first message. You will get the "start" line from the competition port after this thirty seconds and the simulation loop will be started. The stopper starts at this time. You have to drive the car's center to reach the given cell as fast as possible.

3.5.2 Among sheeps

The second competition is done almost perfectly the same way as the previous one, except for the fact that the streets will be full of traffic code obeying cars.

3.5.3 Face to face

In the third challenge two teams are competing. The schedule is the same, except that in the first line after the name of your car, you also get the name of the opponent car. Then the race is going until one of the cars reaches the destination spot.

3.6 Controlling the traffic

Here your task is to create a perfect light controlling scheme for a city. At one point of the competition, a model city becomes available for traffic control. In a round robin system, teams who want to participate can control the lights of this city. After each light controller change, the city is always reset to the same start status, and the cars are always controlled by the same car controllers. Your job is to make the average speed of the cars as high as possible, by switching the lights appropriately. If an accident happens, then you failed your try. Otherwise your result is stored, and the best result of each team will be compared to get the scores.

Chapter 4

Criminals' Handbook

4.1 On the employment system

As described in the Freelance Policemen's Handbook, for years, crime has been a problem in the city. In some years there was too much, in other years too little. Lately the criminal advisor of the mayor has come up with a solution to stabilize the amount of crime. Criminals now become freelance policemen after committing a crime, and they can not commit another until they solve an open police case. This simple rule ensures that there is just as much crime committed as is solved, so all is well in the city.

To ease this process criminal and police employment offices have been set up. Criminals start in the lobby of the employment system. There is a selection of jobs, but only one job can be taken at a time. If the job ends unsuccessfully, the criminal is penalized as described below and is returned to the lobby where they can choose to retry the same job or take a different job.

Once a criminal takes a job, payment depends on two factors. One is successfully completing the job. The other concern is if the identity of the criminal is later revealed by the investigation.

If the job is not completed successfully, the criminal is penalized for \$100 (the account balance can become negative as a result). If it is completed successfully, the criminal is awarded \$50. After a successful job, a police investigation is concluded. The detective can guess multiple times about the identity of the perpetrator. For every mistaken guess, the detective is penalized for \$100. For a correct guess the detective is awarded \$50. (And the investigation ends.) If the very first guess is incorrect, the criminal gets an extra \$50.

Speed is of the essence in these matters, so both the detective and the criminal are awarded money based on the time they took. For all payouts this appears as a percentage. If a job or investigation was executed in under 1 second, the bonus is 100%. Anything slower than 1 second does not receive a bonus.

As an example if a criminal performs a job in 0.2 seconds, and the first guess of the detective is wrong, the criminal is awarded a total \$200.

After a job is completed successfully, the criminal is reassigned to the police department. See the Freelance Policemen's Handbook for details.

4.2 On the jobs

Jobs take place inside buildings. Buildings are made up of tiles on a 2-dimensional grid. People can step on the empty tiles, but only one person fits into an empty tile.

The position of each person can be described by the coordinates of their tile and the direction they are facing.

When you take a job, you receive invaluable briefing about it. The briefing is formatted as follows:

```
JOB CODENAME: %s

JOB DESCRIPTION:

%s

WEAPON RANGE: %d

PEOPLE: %d

MAP SIZE: %d %d

CURRENT POSITION: %d %d
CURRENT FACING: %s
NAME: %s
DESCRIPTION: %s

GOALS:

%s

MAP:

%s
```

`%s` represents a string field in that place and `%d` an integer field.

The codename and description fields are intended to give you a better idea of the job and may hold valuable advice.

Criminals often have to resort to violence, and the weapon range describes its potential. A weapon with the range of 1 is a special hand-to-hand weapon. It can only kill someone standing right in front of the criminal. It is silent and usage is invisible (the witnesses will only be able to tell that the victim died, but not who killed them). Weapons with larger ranges have accordingly increased maximum distances.

The next field gives the count of people in the building, including the criminal.

The size of the map is given with the "width" or "west-east" dimension first and the "height" or "north-south" dimension second.

The position and starting orientation of the criminal is given in building coordinates. X (the first coordinate) starts at 0, meaning the westernmost edge, and increases to the east. Y (the second coordinate) starts at 0, meaning the northernmost edge, and increases to the south. The name is only for reference, but the description can be useful for estimating how easy it will be to identify you from testimonies. (For example if it is "a man", then those who do not know you or see you well will be unable to give a very useful description, but if it is "a robot with red eyes and a green parrot on the shoulder", then you are possibly easier to identify.)

The goals can be of 4 types:

`Visit (%d;%d).`

You have to move to the given coordinates sometime during the mission.

`Avoid (%d;%d).`

You must not move to the given coordinates sometime during the mission.

`%s (who starts at (%d;%d)) must die.`

You must kill this person. The part in parentheses may be omitted in which case you have to find the victim.

`%s (who starts at (%d;%d)) must stay alive.`

You must not kill this person. The part in parentheses may be omitted.

The map is given as rows of characters, each character representing one tile. There are 4 different tiles:

`.`	An empty tile. It can be occupied by a person.
`D`	A door. It can be opened, kicked down or destroyed with a weapon.
`P`	A phone. It can be used to ring other phones or call the police. Can be destroyed with a weapon.
`W`	Wall.

After the briefing the mission starts. It continues until you forfeit or someone calls the police.

During the mission people take turns. They act in the alphabetical order of their names. This means that if the criminal is not the first in order, then others will act earlier at the start of the mission.

4.3 On committing a crime

When the mission is in progress and it is your turn (signaled by an `It is your turn.` message), you have a number of commands at your disposal. These commands can be listed with the `help` command, but are described in greater detail here. Most of the commands take 1 turn to perform. For the few exceptions, the time required is described below.

`forward`	Move to the tile in front of you. Can only be used if the tile is empty.

`turn left`
`turn right`
	Changes the direction you are facing. It not only affects the direction you can move and several other commands, but also your vision. For details see the section on people.

`shoot %d`	Fire your weapon. The number given is the distance to aim at. It must not be greater than the range given in the briefing. You spend distance-1 turns aiming before the shot. In the next turn you shoot. During this time you are visibly armed. The shot is fired straight ahead and will the bullet hit the first thing in its path or drop to the ground when the distance is reached. Phones and doors will become empty tiles if hit and a crash is heard. The police may find the remains of these items in this tile during investigation. People will die if hit. A dead person behaves the same as a living person, except that they can not voluntarily move, call the police or testify. (They still block sight, movement and bullets.)
`call %d`	You can issue this command when standing next to a phone. If the number is 911, you call the police and the mission ends. The detectives will know who has called the police. Any other number is assumed to be the number of a phone in the building. Phones get their numbers based on their coordinates. The X coordinate gives the first four digits and the Y the next four. So a phone at (12;34) would have the number 00120034. If you call a phone in the building it rings. As described later, sounds attract people so it can be used as distraction.
`whistle`	Whistle to attract the attention of those within hearing distance.
`open`	Used to open the door you are facing. It takes two turns. In the first turn you open the door. In the second you move to the position of the door. When you move from this place, the door will automatically close behind you. Opening a door in this way leaves fingerprints which may be found by a careful investigation.
`kick`	Used to destroy the door in front of you. While it does not leave fingerprints, it does make a loud sound and will attract those within hearing distance.
`wait`	Pass your turn.

`drag`	You can grab the person in front of you using this command. The person grabbed can be either dead or alive. If the person is alive, they will be unable to act while they are held. Whether they are alive or not, when they are being dragged, the person will be moved to your previous position when you move with `forward`. (You can not swap places. Turning does not affect the person dragged.) The relative orientation of a person being dragged is fixed, meaning that if you grab someone from behind, no matter how you move them, they will stay facing away from you until you release them. Dragging someone slows you down, so after every `forward` command, you will rest for one turn. If you are dragging someone who is alive, it is not obvious to onlookers. Their testimony will only include that they saw the two of you moving together. (Though if the person you are dragging faces away from you, they will be seen as moving backwards, which does not otherwise occur.) The person being dragged will not be able to tell who dragged them unless they are facing the criminal. Dragging a dead body will leave a trail of blood that may be found during the investigation. It does not take any time to grab someone.
`drop`	Releases the person being dragged. It does not take any time to release someone.
`forfeit`	Ends the job (as unsuccessful).
`quit`	Disconnects. You can reconnect later to continue the job from exactly the same state.
`testify %s`	Adds a line to your testimony. Note that there is no command for removing or changing lines in the testimony, and also that you can not add more lines once the mission is finished.

You have to send these commands terminated by a newline character to the employment server.

Between your turns, you receive messages about what you see. They are self-documenting, but should still not be difficult to parse. For example, the message `You see John at (5;4) facing north.` means that you see the person named John at coordinates (5;4) facing north. These messages are sent each on their own line. These are the same messages that will be included in testimonies, except that they start with "You" instead of "I". For example `I see John at (5;4) facing north.` could be a line in a testimony (they are in present tense as they are reciting their memory).

Not all actions are visible to observers. The visible actions include `turn`, `forward`, `whistle`, `kick`, `open` and `shoot` (unless the weapon range is 1). Dying is also visible.

4.4 On people

The buildings that jobs take place are most often populated with people. Understanding their behavior is key to being a successful criminal.

4.4.1 Movement rules

Rule 1. People start facing north.

Rule 2. People have a starting position and possibly a list of positions that make up their route. If they have a route, they move to the first point in the route, then the second, and so on until the last, after which they start over and move to the first position again.

Rule 3. People have a complete intuitive knowledge about the building. They do not only know about the initial configuration, but about any changes you may make (such as doors kicked down and phones destroyed), and also about the positions of everyone. This knowledge is not conscious (it will not be reflected in their testimony), but they use it for finding the shortest path wherever they are going.

Rule 4. When a person is going somewhere (such as the next position in their route), they find the shortest path to the destination and perform the first move (`turn` or `forward` action). In the next turn they will find the shortest path from their new situation, so they can react to changes in the map dynamically.

Rule 5. If there is no path to their destination they will not move. They will wait until a path becomes available or their destination changes. (Note that for example if you whistle and someone hears it, their destination will become your position. Since it is blocked – you are standing there –, they will not start moving. When you leave that tile however, they will get started.)

Rule 6. When searching for a path, they look for the shortest path in terms of turns needed (counting `turn` and `forward` actions). They look for the shortest path favoring `forward` over `turn` actions and `turn left` over `turn right`.

Rule 7. People other than the criminal do not open doors and will only go through a door if it is destroyed or if they are dragged through it.

4.4.2 Sensory rules

Rule 1. People can hear sounds from 10 tiles away. The distance is calculated as the sum of the north-south and east-west distances. When they hear a sound, people will decide to investigate its source (by going there and then resuming their route). Since the criminal is the cause of all sounds, they do not get notified about them.

Rule 2. People can see 10 tiles away. The distance is calculated as the sum of the north-south and east-west distances. From a distance of 10 tiles they can only give vague descriptions, but from 5 tiles, they recognize people that they know. Acquaintance is symmetric – if (and only if) you know a person do they know you too. The criminal always knows those marked for eliminations.

Rule 3. People have a 90 degree arc of vision. In the figure below, empty tiles in the arc of vision of a person at `@` facing south are marked with `+`.

```
WWWWWWWW
W......W
W...@..W
W..+++.W
W.+++++W
W++++++W
W++++++W
WWWWWWWW
```

Rule 4. A person can see another tile, if the line connecting the central points of the two tiles is not interrupted by a non-empty tile (a tile containing a door, phone or wall or another person). If just the corner of a tile touches this line of vision, then it is not interrupted.

In the figure below, empty tiles in the field of vision of a person at `@` facing south are marked with `+`.

```
WWWWWWWWWW
W.....@..W
W....+++.W
W...+++++W
W..++W+++W
W.++..+++W
W+++..+++W
W++...+++W
W+...++++W
WWWWWWWWWW
```

Rule 5. Auditory and visual perceptions will be included in testimonies, as well as an account of what the witness did. The exact syntax has to be learned as a detective.

Rule 6. Gunshot sounds are special. They can be heard in the whole building, and people do not wish to investigate it (they prefer to live). The testimony for gunshot sounds includes a timestamp, that is based on a random number plus the Euclidean distance from the source of the sound. Detectives can use this information for locating the shooter. The criminal also gets notified about the gunshot sound, so that they learn the random number.

Rule 7. If a person sees a dead body or an armed person they will panic. A panicked person will move to the nearest phone and call 911 when they get there. (They actually look for the closest – in terms of Movement Rule 6. – empty tile next to a phone.)

Chapter 5

Freelance Policemen's Handbook

5.1 On the employment system

As described in the Criminals' Handbook, for years, crime has been a problem in the city. In some years there was too much, in other years too little. Lately the criminal advisor of the mayor has come up with a solution to stabilize the amount of crime. Criminals now become freelance policemen after committing a crime, and they can not commit another until they solve an open police case. This simple rule ensures that there is just as much crime committed as is solved, so all is well in the city.

To ease this process criminal and police employment offices have been set up. Everyone starts as a criminal, but once they have successfully committed a crime, they are transferred to the employment of the police department. These freelance policemen are then placed in a queue. (At this time they receive the message `You are waiting in the freelance police queue.`.) When a crime is committed then, the police office looks at the queue and assigns it to the first policeman eligible. Everyone is eligible who has not done either of:

- Taking the job as a criminal. (Either completing it successfully or unsuccessfully.)
- Solving the case as a policeman. (Unsuccessfully guessing does not void eligibility.)

When a policeman is assigned a job, they get the message `You have received an assignment from the commissioner!` and a report from the crime scene investigators containing every relevant data. They have the option to "dust" a doorhandle to see if there are any fingerprints on it. This can only be done once before guessing at the identity of the criminal. If the guess is incorrect, they are placed at the end of the queue and penalized for \$100. If the guess is correct, they are awarded \$50 (or \$100 if the guess was submitted within 1 second of the assignment).

After a correct guess the policeman is once again discharged and reassigned to the criminal employment system. For further details see the Criminals' Handbook.

On the cases

Cases describe crimes committed inside buildings. Buildings are made up of tiles on a 2-dimensional grid. People can step on the empty tiles, but only one person fits into an empty tile.

The position of each person can be described by the coordinates of their tile and the direction they are facing.

When you are assigned a case, you receive the report of the crime scene investigators. The report is formatted as follows:

```
CASE CODENAME: %s

PEOPLE:
```

```
%s

POLICE WAS CALLED BY: %s

TESTIMONIES:

%s

CLUES FOUND:

%s

MAP SIZE: %d %d

MAP:

%s
```

`%s` represents a string field in that place and `%d` an integer field.

The codename is just used to identify the case for humans.

People are described on separate lines in the following format:

```
%s: %s at (%d;%d) facing %s, %s
```

The variables substituted in the format string are in sequence:

- the name of the person (a single word, and nobody has the name "TESTIMONIES")
- a description of the person, such as `an old man in a blue jacket`
- the X coordinate (starts at 0 on the westernmost edge of the map and increases to the east)
- the Y coordinate (starts at 0 on the northernmost edge of the map and increases to the south)
- the direction the person is facing
- either `dead` or `alive`, describing the condition of the person.

These describe the people at the moment someone called 911. The next field in the case report gives the name of the person who called 911.

The next section contains the testimonies of every living person. Testimonies start with `%s has remembered the following:` and end with `And that is all %s has remembered.` (both fields are filled in with the name of the person). Testimonies are separated with an empty line, and between their beginning and end contain the memories of the witness, each memory on a separate line. These lines are self-documenting, but should still not be difficult to parse. For example, the memory `I see John at (5;4) facing north.` means that they saw the person named John at coordinates (5;4) facing north. (They are in present tense as the witness is reciting their memory.) The memories are listed in the order they were experienced, but are not timestamped.

The following section lists the clues that were found by the crime scene investigators on separate lines. The possible clues are:

- `Found some blood at (%d;%d).` (Someone was killed there or a dead body was dragged there.)
- `Found a bullet casing at (%d;%d).` (As criminals are right-handed, the bullet casing flies off from the rifle to the right when they shoot. If the tile to the right of the criminal is empty, the casing will land on this tile. If this tile is not empty, the casing bounces back to the feet of the criminal and lands on his tile.)

- `Found a destroyed door at (%d;%d).` (Kicked in or shot.)
- `Found a destroyed phone at (%d;%d).` (Shot.)
- `Found a bullet at (%d;%d).` (The bullet hit the wall or hit the ground after traveling for the set distance.)

The two numbers at the ends of the lines give the coordinates the clue was found at. Consulting the Criminals' Handbook should give a solid idea about what these clues might indicate.

The size of the map is given with the "width" or "west-east" dimension first and the "height" or "north-south" dimension second.

The map is given as rows of characters, each character representing one tile. There are 4 different tiles:

`.`	An empty tile. It can be occupied by a person.
`D`	A door. It can be opened, kicked down or destroyed with a weapon.
`P`	A phone. It can be used to ring other phones or call the police. Can be destroyed with a weapon.
`W`	Wall.

For more documentation on how the world has operated before the police was called see the Criminals' Handbook.

Before submitting a guess at the perpetrators identity with the `execute %s` command (with the suspect's name), detectives can use the `dust (%d;%d)` command with the coordinates of a door. It will either result in `No fingerprints found on door handle at (%d;%d).` or `The fingerprints of %s have been found on door handle at (%d;%d).` with the name of a person who has left their fingerprints on the door handle. The `dust` command may only be used once before a guess.