Checkpoint City

Project Report

Marco Bouterse
1142828
mcbouter@cs.vu.nl

Multimedia Authoring II
Vrije Universiteit, September 2006

www.cs.vu.nl/~mcbouter/mma2

Introduction

Multimedia Authoring II is about creating interactive web experiences using Distributed Logic Programming (DLP) and the Virtual Reality Modeling Language (VRML). DLP offers object-oriented parallel logic programming and is able to manipulate objects in a VRML world. This report describes my final project for this course, a checkpoint racing game that can be played in a web-browser using the mentioned techniques. First the system design will be presented, followed by a technical description of the implementation. Then a user manual is presented that explains all controls and features of the game. Finally a discussion of the solution and possible extensions will be presented.

Contents

Game Design

General Overview

Checkpoint City is an arcade racing game implemented in DLP and VRML. The player controls a car driving through a virtual city. The goal is to find the next checkpoint before a timer runs out. When the player reaches the checkpoint in time, a new one appears somewhere in the city and the timer is increased. The player wins when all checkpoints have been reached in time. Traffic prevents the player from reaching the checkpoints easily.

Target System & Requirements

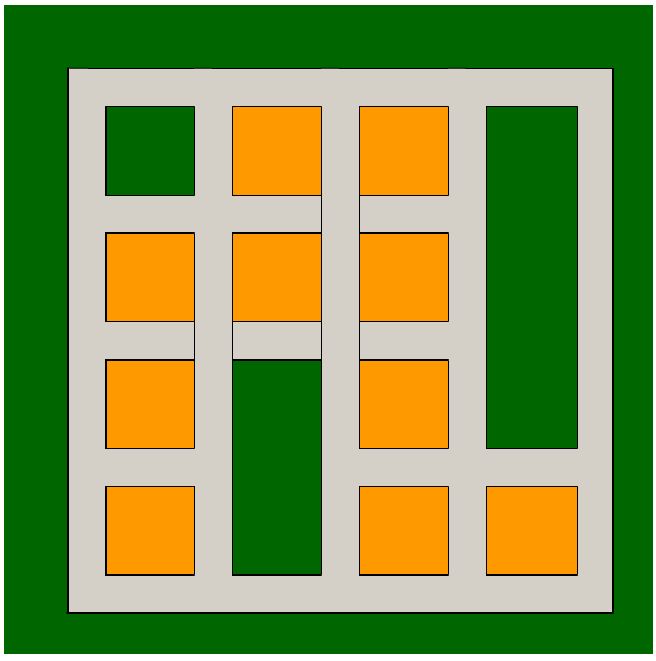
The game should be playable with a standard Windows computer that is not too old (performance will suffer from slow processors). To play the game Internet Explorer with the Blaxxun plugin (www.blaxxun.com) is required.

Playing a game

The player will start a new game by selecting 'Start Game' from the menu (if available, otherwise the game will start automatically when finished loading the program). He can control a car that drives to the streets of a virtual city. The car drives automatically, the player can control it by braking, changing lane or turning at a crossing (using the arrow keys on the keyboard). The goal is to find the checkpoint before a timer runs out. If the player drives the car through the checkpoint, the timer is increased with a certain amount of time and another checkpoint appears somewhere in the city. The game is finished when the timer runs out, the player hits another car or if all checkpoints are reached (player wins).

World Design

The virtual world that the game is situated in represents a city with streets. These streets are organized as a 5x5 grid. The nodes in this grid are the crossings and the edges are roads. The figure on the left shows a grid based street layout. There are only straight streets, because curved streets introduce a lot of extra problems that cannot be dealt with in the scope of this project. The space between the roads is filled with either buildings or parks. The design in the picture also shows elevation differences between roads, introducing bridges and/or tunnels. On crossings with roads coming in from all directions there should be traffic lights that are either operated on a time based cycle or act more intelligently by detecting whether there is a car waiting and turn to



green if there aren't any cars waiting for the other directions (first come, first served basis).

System Design

All visible elements (city buildings, roads, cars, etc.) will be created using VRML. The gameplay is partly implemented in JavaScript nodes embedded in the VRML code and partly in DLP. To ensure smooth movement, the forward movement of the player car will be implemented in VRML. Most of the game logic, the car controls and the traffic cars will be controlled by DLP code.

Targets

To be successful the game should have the following properties:

1. **Smooth driving experience**, player must be "in control".
2. **Collision Detection**, to detect when hitting checkpoints and other cars.
3. **Autonomous cars**, computer guided traffic must drive believable.
4. **Solid gameplay**, game must be challenging and fun.

Technical Description

This part gives a technical overview of the game. It is split into two main parts: the VRML part and the DLP part. The VRML part describes all objects and their properties and scripts. The DLP part describes all DLP objects that control elements in the game.

VRML Objects

These objects can be found at www.cs.vu.nl/~mcbouter/mma2/vrml/city/prototypes/

Checkpoint.wrl

This file contains the prototype for the checkpoint object, consisting of a 3D letter "C", enclosed by a transparent shape. The checkpoint is rotated along its Y-axis by using interpolators and ROUTE statements. This object was created using Flux studio.

City.wrl

This file contains the main VRML code that is loaded into the browser by the DLP code. It puts all other pieces together resulting in a virtual city. It starts by loading all external prototypes from the other files. A skybox is set up that represents the sky by night. Using the road pieces from road_protos.wrl the city roads are pieced together like a jigsaw puzzle. A five by five grid is created and the spaces between are filled by parks and skyscrapers. As a finishing touch, the surrounding terrain is put into place and a couple of windmill objects are placed in it. Next the objects that are manipulated by DLP code are declared: the checkpoint, the player's car and the other traffic. The file ends with two scripts. The first one implements the timer and decrements it every second. If the timer drops below 11, the time left will be shown in a red color. If the timer reaches 0, the sentence "GAME OVER" will be displayed. The second script handles the input and forward movement of the player's car. Every 1/100 of a second the script is invoked and calls the moveCar() function that calculates the car's current speed and forwards the car in the current direction. The actionKeyPressed() function updates the current input status (left, right & down key). A complete listing of this file is provided in appendix A.

Park.wrl

Contains the prototype for the park areas. Basically just a Box shape with a grass texture applied to it. This object was created by hand using an editor.

Player_car.wrl

Contains the car model prototype that is used for all cars in the game. It also contains a Wheel prototype that is used to represent the wheels of the car. Using some code the wheels are rotating as if the car is driving. It is also possible to change the steering angle of the front wheels through the prototype interface. Also the emissive and diffuse colors of the car can be changed through its interface. The object is based on a vrml1 model found on the internet. I converted it to vrml2 using 3d studio max and made it into a prototype, added wheel spin, wheel steering, gave it new colors and added color control.

Road_protos.wrl

This file contains the basic road pieces for building a city, including objects that are found on these pieces. It contains prototypes for a traffic light (created from scratch using Flux Studio), a street lamp (based on existing model) and road pieces that can contain these objects. RoadStraight contains a straight road with sidewalks. RoadStraight2 is almost the same, but also has a street lamp on one side of the road that is activated if the player is nearby. RoadBend is a sort of L shaped piece that can be used for corners. RoadT contains a T-section with roads coming in from 3 sides. The final piece is RoadCross and contains a full crossing with 4 incoming roads including traffic lights and a script that operates the traffic lights based on a 20 second time interval. Opposing traffic lights are green at the same time, then go to orange and finally to red. At that point the other two traffic lights will switch to green, etc.

Skyscraper.wrl

Contains the prototype for a skyscraper model that is converted from an existing vrml1 model. The colors have been adapted to fit the mood of the game better.

Terrain.wrl

Holds the prototype for the terrain that surrounds the city limits. Created using a text editor. It contains a couple of textured ElevationGrid nodes that define the terrain's height at several points.

Windmill.wrl

This file contains the prototype for a windmill that is created by converting an existing vrml1 model. The model is enhanced by adding code to animate the windmill.

DLP Objects

These objects are found in the file ccity.pl: a listing of this file is found in appendix B.

Ccity object

This is the main object that loads the world into the browser, sets up the text area for debugging purposes and then creates different threads for the game objects. First five traffic threads are started, that each manage a single car traveling randomly through the city. Next the checkpoint thread is started, that takes care of displaying the checkpoints and checks if the player drives through them. It also increases the timer that is maintained in the city.wrl file. Finally the player_car thread is started that manages the player's car and the user input.

Player_car object

The constructor clause of *player_car/0* activates the TimeSensor ts1 in city.wrl, causing the car to start moving forward (regardless which direction it is traveling in). It then starts the main loop clause: *update/0*.
Update/0 is an endless loop that checks the position and orientation of the car, it then calculates the relative position of the car (using modulo) to be able to determine if the car is on a crossing or not. Also the coordinates of the closest crossing are calculated, so if the car is actually on the crossing, the game knows which crossing it is on. It

then executes the *decide_action/6* clause to determine which action should be taken. The possible actions are:

- keep_going: Just let the car keep going forward
- turn_right: Make the car turn right on a crossing
- turn_left: Make the car turn left on a crossing

The chosen action is then executed by the *do_action/2* clause.
Decide_action/6 takes as parameters the car's orientation, the relative position and the nearest crossing's coordinates. It returns the action to follow. Basically there are 6 different situations:

1. The car is **not** on a crossing: just let the car go forward, return action is keep_going. In all following situations, the car is on a crossing.
2. If the right arrow key is pressed, check if there is a road to the right (using the *connected* clause). If there is a road, rotate the car 90 degrees to the right. Return action is turn_right.
3. If the left arrow key is pressed, check if there is a road to the left (using the *connected* clause). If there is a road, rotate the car 90 degrees to the left. Return action is turn_left.
4. If there is no input, check if there is a road on the other side of the crossing, if so make the car keep going straight. Return action is keep_going.
5. If there is no input and it is not possible to go forward, check if there is a road to the right, if so return action will be turn_right.
6. If there is no input and it is not possible to go forward or right, check if there is a road to the left, if so return action will be turn_left.
7. The above situations should cover all possible events (except dead-end streets, but they should not be there). If none of the above succeeds, the catch-all clause will return no_action. This is considered as an error and should not happen.

The *do_action/2* clause is responsible for executing the action. It takes as parameters the car's orientation and the action to be taken. If the action is keep_going, nothing is done, because the car will automatically be forwarded by the VRML script in city.wrl. If the action is turn_left or turn_right, the car is rotated 90 degrees in the proper direction, followed by a 1 second pause (to prevent the car from multiple rotations on a crossing).
The *connected/2* clause is used to find out connections between crossings. Two crossings A and B are connected if there is a road from A to B or from B to A. The roads are represented by a database of *road* clauses. The crossings are identified by a [X,Z] pair, [1,1] being the top-left crossing (if viewed from above and VRML world coordinates (0,0) are at top left. [5.5] is then the bottom right crossing. Also see the DLP source for a schematic representation. *Road([X1,Z1],[X2,Z2])* means that there is a road from crossing [X1,Z1] to crossing [X2,Z2].

Checkpoint object

The constructor starts with enabling the timer in city.wrl, which causes the timer to start counting down on screen. Then the *game_loop/1* clause is started.
The *game_loop/1* clause takes the current checkpoint id as a parameter. It looks up that specific checkpoint in a database and retrieves the info: position in X, Y and Z coordinates and the number of seconds that the player has to find that checkpoint. Using this info, the checkpoint is positioned at the retrieved coordinates and the timer is increased. Then the wait for the player to hit the checkpoint begins, which is handled by the *check_collision/3* clause. If there has been a collision, the checkpoint

id is decremented and *game_loop/1* is recursively called using the new id. If the id hits 0, the player has found all checkpoints and this is communicated to the VRML script in city.wrl, which displays the final score for the player. The score equals the number of seconds still left when the player hits the last checkpoint.
The clause *check_collision/3* loops forever until a collision is detected between the car and the checkpoint. This uses a fairly simple approach by monitoring the distance between the center of the checkpoint and the center of the car. If this drops below a certain threshold value, a collision is detected and the *game_loop/1* can continue.
To calculate the distance, the *distance/5* clause is used. It uses Pythagoras to calculate the distance between two points.

Traffic object

The constructor *traffic/2* takes as parameters the name of the VRML object (in this case a car) and the position to start from. These starting positions are very specific: they should be on the right lane exactly one unit on a crossing. If the starting position is wrong the car won't stay on the roads all the time. The constructor starts by positioning the car at the start position, then calculates the crossing coordinates that the car is on and finally starts the *traffic_loop/3*.
The *traffic_loop/3* clause takes as parameters the object name and the current crossing coordinates. It uses the *random/1* clause to retrieve a set of destination coordinates. Next a route from the current location to the destination is calculated using the *route/3* clause. Then this route is being passed to the *drive_to_destination/2* clause that takes car of driving the car to the new destination. At the destination the car waits for 5 seconds and then the *traffic_loop/3* is executed again to drive to the next destination.
Drive_to_destination/2 takes as parameters the object name and a route, which is a list of [X,Z] pairs representing connected crossings that can be used to drive from the current location to the destination. The first two pairs from this list are extracted. The first pair represents the current crossing, the second pair represents an adjacent crossing that must be traveled to in order to reach the final destination. To obtain the action that must be taken from the current to the next crossing, *decide_action/4* is used, which will return one of the following actions: go_straight, turn_right, turn_left or turn_back. The clause *do_action/4* takes care of executing each of these actions and after it is finished, the car will be at the next crossing. The first element of the list is no longer needed. The second element is now the current location and is passed along with the rest of the route to the *drive_to_destination/2* clause again. The stop condition (to break the recursion) is when the list only contains 1 element, namely the current location. In this case the destination has been reached.
Decide_action/4 just compares current location and destination based on current orientation to decide whether to go straight, turn to the left or right or go back. The clause *do_action/4* takes care of the needed animation and moves the car from the current crossing to the next one possibly rounding a corner in the process. It uses *move_to_position/4* (interpolates between two points) to do this.
The last important clause of the traffic object is *route/3*. This clause takes as parameters the coordinates of the current crossing, coordinates of the destination crossing and returns a route between these crossings as a list of [X,Z] pairs, representing the crossings that must be passed to reach the destination. It doesn't use a shortest path algorithm, but it keeps a list with intermediate crossings and doesn't visit the same crossing more than once to prevent getting stuck in cycles. Finally the traffic object has its own private copy of the road database and the *connected/2* clause.

User Manual

Starting a game

To start a new game, open a new browser window of Internet Explorer and enter the URL http://www.cs.vu.nl/~mcbouter/mma2. Look for 'Final Project' and click on the link named 'Checkpoint City'. The game will now be loaded into the browser window. As soon as the world has been loaded the game will start automatically. If the car starts to drive forward, the game has begun. To make sure that the key presses are detected click twice in the displayed area. The game can also be started by directly entering the full URL:
http://www.cs.vu.nl/~mcbouter/mma2/exercises/checkpoint_city.html

Playing a game

The object of the game is to drive through the virtual city searching for checkpoints. The checkpoint has to be found before the timer that is displayed on screen runs down to zero. If the checkpoint has been found and hit (by driving through/along it). The timer is increased by a certain amount and the checkpoint moves to another location. There are two possible endings to the game. If the timer reaches zero, the game is over and the player has lost. If all checkpoints are found before the time runs out, the player has won. The player scores points if he has any seconds left on the timer.

Controls

The controls are very simple. The car is moved forward automatically if no input is given. If the car is about to drive off the road, it will be automatically turned in a possible direction by the game. To make the car turn left, press and hold the left key when reaching a crossing. After the car has turned, the key can be released. The same with the right key for making a right turn. It is not possible to turn off road. Besides the left and right arrow key, the only other key that can be used is the down arrow key, which acts as a brake. To make finding the checkpoint easier a 'MAP' button is available on the lower right corner of the screen. Click and hold to change the perspective to a top-down view that looks over the entire city. Releasing the left mouse button will return to the default view (behind the car).

Known Bugs

- As for now, there is no collision detection between cars driving through the city and between the player's car and those cars.
- Sometimes the control seems to be lost and the player's car just drives straight on off road. This is very annoying and lots of time has been spent to find out what goes wrong. No solution has been found yet.
- Sometimes not all textures are loaded into the VRML world.
- Sometimes a key press is not detected properly, so the car doesn't turn.

Discussion & Future Work

Development Process

Looking back at my original design and expectations, I must conclude that they were a little too ambitious. I imagined a completely drivable car including gas/brake and steering that could move freely through a city, reacting to the environment realistically (crashes with buildings, reacting to curbs etc.). The fact that VRML doesn't really have a good input system that allows for detecting multiple key presses at the same time (gas and steer for example) prevented this approach. In the original design there were also elevation differences (resulting in bridges), something that also appeared to be hard to implement. Finally the independent car agents that waited for traffic lights, minded each other and tried to avoid crashes were not entirely feasible in the given amount of time. A lot of time has been spend on finding, improving and creating VRML models to put together a nice city environment. Programming the game logic using Distributed Logic Programming was also a very time-consuming job. This was due to the fact that my Prolog foundation wasn't as rock solid as it could have been, resulting in many time spent on searching for certain Prolog features or finding out how to do something that is trivial in a procedural language in logic programming. Many hours have been spent trying to find and fix the bug that causes the controls to fail at random moments. I did not succeed in this and I still don't know why this is happening. A possible solution seems to be to let DLP handle the car's forward motion, but this doesn't look good at all (motion is not smooth). Also built-in fixes to reset the car once it strays off course do not seem to work. It looks like once the error occurs, the complete DLP car thread fails to continue working.
Also the fact that there aren't a lot of tools and support for DLP programming did not contribute to a very smooth development process. Having the summer vacation in between the creation of the final project was an obstacle too that did not improve the continuity of the project.
After all I ended up with a game that works and has reasonable gameplay, although I could have used a little more time to fine-tune everything and really create a game that is challenging and fun to play.

Possible Extensions

If I had more time to work on this project I would start with getting the collision detection system working, so the traffic in the city has a real use. I would also extend the control over the player's car so that it stays in one lane and can change between lanes on straight roads (so the traffic can be evaded safely). The traffic agents can also be extended to become 'smarter' so they would stop for traffic lights and other cars. This would really improve the overall gameplay. After that some 'fancy' features can be introduced like pedestrian agents, a menu structure (no need to restart browser), an introduction on first startup, background music etc. The city can also be bigger than it is now (this is actually very easy to realize, because all that is needed are new roads in VRML that follow the grid layout and new road/2 statements in the DLP code).

References

- The Annotated VRML 2.0 Reference Manual, Rick Carey and Gavin Bell, Addison Wesley, 1997.
- Blaxxun Platform 7.0 Documentation [http://developer.blaxxun.com/doc/wwhelp/js/html/frames.htm]
- Distributed Logic Programming for Virtual Environments, Zhisheng Huang, Anton Eliëns and Cees Visser
- PROLOG Built-in Documentation (help system)
- DLP Release Notes [http://dlp.cs.vu.nl/~ctv/dlpinfo/docs/readme.html]
- WASP Project Homepage [http://wasp.cs.vu.nl/wasp/]

Appendix A – VRML source for city.wrl

```
#VRML V2.0 utf8

EXTERNPROTO       RoadStraight [
    exposedField SFVec3f translation
    exposedField SFRotation rotation
    exposedField SFVec3f scale
    exposedField MFString roadURL
    exposedField MFString sidewalkURL
]
"road_protos.wrl#RoadStraight"

EXTERNPROTO       RoadStraight2 [
    exposedField SFVec3f translation
    exposedField SFRotation rotation
    exposedField SFVec3f scale
    exposedField MFString roadURL
    exposedField MFString sidewalkURL
]
"road_protos.wrl#RoadStraight2"

EXTERNPROTO       RoadBend [
    exposedField SFVec3f translation
    exposedField SFRotation rotation
    exposedField SFVec3f scale
    exposedField MFString roadURL
    exposedField MFString sidewalkURL
]
"road_protos.wrl#RoadBend"

EXTERNPROTO       RoadT [
    exposedField SFVec3f translation
    exposedField SFRotation rotation
    exposedField SFVec3f scale
    exposedField MFString roadURL
    exposedField MFString sidewalkURL
]
"road_protos.wrl#RoadT"

EXTERNPROTO       RoadCross [
    exposedField SFVec3f translation
    exposedField SFRotation rotation
    exposedField SFVec3f scale
    exposedField MFString roadURL
    exposedField MFString sidewalkURL
]
"road_protos.wrl#RoadCross"

EXTERNPROTO       Car [
    exposedField SFVec3f translation
    exposedField SFRotation rotation
    exposedField SFVec3f scale
    exposedField SFBool bindCarView
    exposedField SFRotation wheelRotation
    exposedField SFColor carDiffuseColor
    exposedField SFColor carEmissiveColor
]
"player_car.wrl#Car"

EXTERNPROTO Park [
    exposedField SFVec3f translation
    exposedField SFRotation rotation
    exposedField SFVec3f scale
    exposedField MFString grassURL
]
"park.wrl"

EXTERNPROTO SkyScraper [
    exposedField SFVec3f translation
    exposedField SFRotation rotation
    exposedField SFVec3f scale
]
```

```
"skyscraper.wrl"

EXTERNPROTO CheckPoint [
    exposedField SFVec3f translation
    exposedField SFRotation rotation
    exposedField SFVec3f scale
]
"checkpoint.wrl"

EXTERNPROTO Terrain []
"terrain.wrl"

EXTERNPROTO Windmill [
    exposedField SFVec3f translation
    exposedField SFRotation rotation
    exposedField SFVec3f scale
]
"windmill.wrl"


EXTERNPROTO KeySensor[
    eventIn SFBool eventsProcessed
    exposedField SFBool enabled
    eventOut SFInt32 keyPress
    eventOut SFInt32 keyRelease
    eventOut SFInt32 actionKeyPress
    eventOut SFInt32 actionKeyRelease
    eventOut SFBool shiftKey_changed
    eventOut SFBool controlKey_changed
    eventOut SFBool altKey_changed
    eventOut SFBool isActive
]["urn:inet:blaxxun.com:node:KeySensor",
"http://www.blaxxun.com/vrml/protos/nodes.wrl#KeySensor"]

EXTERNPROTO MouseSensor[
    eventIn      SFBool      eventsProcessed
    exposedField SFBool      enabled
    eventOut     SFVec2f     client
    eventOut     SFVec2f     position
    eventOut     SFBool      lButton
    eventOut     SFBool      mButton
    eventOut     SFBool      rButton
    eventOut     SFFloat     mouseWheel
    eventOut     SFBool      isActive
    ]
["urn:inet:blaxxun.com:node:MouseSensor",
"http://www.blaxxun.com/vrml/protos/nodes.wrl#MouseSensor",
]

# Type should be "NONE" for gameplay
DEF NI1 NavigationInfo {
    type "NONE"
    headlight FALSE
}

PointLight {
    ambientIntensity .5
    attenuation 1 0 0
    color .6 .6 .6
    intensity .5
    location 85 10 85
    radius 120
}



Background {
    backUrl        "../textures/space_1_back.jpg"
    leftUrl        "../textures/space_1_left.jpg"
    frontUrl       "../textures/space_1_front.jpg"
    rightUrl       "../textures/space_1_right.jpg"
    bottomUrl      "../textures/space_1_bottom.jpg"
    topUrl         "../textures/space_1_top.jpg"
}

#Fog {color .7 .9 .7 visibilityRange 600}
```

```
DEF INPUT_SENSOR KeySensor {}
DEF MOUSE_SENSOR MouseSensor {}

# User Interface
Layer2D {
  children [
    Transform {
      translation 0 .3 0
      children [

            Shape {
              appearance Appearance {
                    material DEF TXTMAT Material {
                      diffuseColor 0 0 0
                      specularColor 0 0 0
                      emissiveColor 1 1 1
                      ambientIntensity 0
                      transparency 0.3
                    }
              }
              geometry DEF TEXT1 Text {
                    fontStyle FontStyle   {
                      family ["Times New Roman","Arial"]
                      justify "MIDDLE"
                      size 0.11
                  }
                    string ["Welcome","to","Checkpoint City"]
              }
            }
      ]
    }
    Transform {
      translation .8 -.6 0
      children [
            DEF TS1 TouchSensor {}
            Shape {
              appearance Appearance {
                    material Material {
                      diffuseColor 0 0 0
                      specularColor 0 0 0
                      emissiveColor .7 .5 .5
                      ambientIntensity 0
                      transparency 0.4
                    }
              }
              geometry Box {
                    size .2 .2 1
              }
            }
            Shape {
              appearance Appearance {
                    material Material {
                      diffuseColor 0 0 0
                      specularColor 0 0 0
                      emissiveColor 1 1 0
                      ambientIntensity 0
                      transparency 0.1
                    }
              }
              geometry Text {
                    fontStyle FontStyle   {
                      family ["Arial"]
                      justify "MIDDLE"
                      size 0.1
                  }
                    string ["MAP"]
              }
            }
      ]
    }
  ]
}

#Sound effects
Sound {
    location 85 1 85
    maxBack 500
```

```
    maxFront 500
    minBack 200
    minFront 200
    spatialize FALSE
    intensity 1
    source DEF AC1 AudioClip {
           url "../sound/applause.wav"
           loop FALSE
    }
}
Sound {
    location 85 1 85
    maxBack 500
    maxFront 500
    minBack 200
    minFront 200
    spatialize FALSE
    intensity 1
    source DEF cpSound AudioClip {
           url "../sound/chimes.wav"
           loop FALSE
    }
}
Sound {
    location 85 1 85
    maxBack 500
    maxFront 500
    minBack 200
    minFront 200
    spatialize FALSE
    intensity 1
    source DEF AC2 AudioClip {
           url "../sound/carbrake.wav"
           loop FALSE
    }
}

# City roads
Transform {
    scale 2 2 2
    children [

RoadBend          {translation 2.5 0 2.5}
RoadStraight      {translation 2.5 0 7.5}
RoadStraight2     {translation 2.5 0 12.5}
RoadStraight      {translation 2.5 0 17.5}
RoadT                   {translation 2.5 0 22.5 rotation 0 1 0 1.571}
RoadStraight      {translation 2.5 0 27.5}
RoadStraight2     {translation 2.5 0 32.5}
RoadStraight      {translation 2.5 0 37.5}
RoadT                   {translation 2.5 0 42.5 rotation 0 1 0 1.571}
RoadStraight      {translation 2.5 0 47.5}
RoadStraight2     {translation 2.5 0 52.5}
RoadStraight      {translation 2.5 0 57.5}
RoadStraight      {translation 2.5 0 62.5}
RoadStraight      {translation 2.5 0 67.5}
RoadStraight2     {translation 2.5 0 72.5}
RoadStraight      {translation 2.5 0 77.5}
RoadBend          {translation 2.5 0 82.5 rotation 0 1 0 1.571}

RoadStraight      {translation 7.5 0  2.5 rotation 0 1 0 1.571}
RoadStraight      {translation 7.5 0 22.5 rotation 0 1 0 1.571}
RoadStraight      {translation 7.5 0 42.5 rotation 0 1 0 1.571}
RoadStraight      {translation 7.5 0 82.5 rotation 0 1 0 1.571}

RoadStraight2     {translation 12.5 0  2.5 rotation 0 1 0 1.571}
RoadStraight2     {translation 12.5 0 22.5 rotation 0 1 0 1.571}
RoadStraight2     {translation 12.5 0 42.5 rotation 0 1 0 1.571}
RoadStraight2     {translation 12.5 0 82.5 rotation 0 1 0 1.571}

RoadStraight      {translation 17.5 0  2.5 rotation 0 1 0 1.571}
RoadStraight      {translation 17.5 0 22.5 rotation 0 1 0 1.571}
RoadStraight      {translation 17.5 0 42.5 rotation 0 1 0 1.571}
RoadStraight      {translation 17.5 0 82.5 rotation 0 1 0 1.571}

RoadT                   {translation 22.5 0 2.5}
RoadStraight      {translation 22.5 0 7.5}
```

```
RoadStraight2     {translation 22.5 0 12.5}
RoadStraight      {translation 22.5 0 17.5}
RoadCross         {translation 22.5 0 22.5}
RoadStraight      {translation 22.5 0 27.5}
RoadStraight2     {translation 22.5 0 32.5}
RoadStraight      {translation 22.5 0 37.5}
RoadCross         {translation 22.5 0 42.5}
RoadStraight      {translation 22.5 0 47.5}
RoadStraight2     {translation 22.5 0 52.5}
RoadStraight      {translation 22.5 0 57.5}
RoadBend          {translation 22.5 0 62.5 rotation 0 1 0 1.571}
RoadStraight      {translation 22.5 0 82.5 rotation 0 1 0 1.571}

RoadStraight      {translation 27.5 0  2.5 rotation 0 1 0 1.571}
RoadStraight      {translation 27.5 0 22.5 rotation 0 1 0 1.571}
RoadStraight      {translation 27.5 0 42.5 rotation 0 1 0 1.571}
RoadStraight      {translation 27.5 0 62.5 rotation 0 1 0 1.571}
RoadStraight      {translation 27.5 0 82.5 rotation 0 1 0 1.571}

RoadStraight2     {translation 32.5 0  2.5 rotation 0 1 0 1.571}
RoadStraight2     {translation 32.5 0 22.5 rotation 0 1 0 1.571}
RoadStraight2     {translation 32.5 0 42.5 rotation 0 1 0 1.571}
RoadStraight2     {translation 32.5 0 62.5 rotation 0 1 0 1.571}
RoadStraight2     {translation 32.5 0 82.5 rotation 0 1 0 1.571}

RoadStraight      {translation 37.5 0  2.5 rotation 0 1 0 1.571}
RoadStraight      {translation 37.5 0 22.5 rotation 0 1 0 1.571}
RoadStraight      {translation 37.5 0 42.5 rotation 0 1 0 1.571}
RoadStraight      {translation 37.5 0 62.5 rotation 0 1 0 1.571}
RoadStraight      {translation 37.5 0 82.5 rotation 0 1 0 1.571}

RoadT                    {translation 42.5 0 2.5}
RoadStraight      {translation 42.5 0 7.5}
RoadStraight2     {translation 42.5 0 12.5}
RoadStraight      {translation 42.5 0 17.5}
RoadCross         {translation 42.5 0 22.5}
RoadStraight      {translation 42.5 0 27.5}
RoadStraight2     {translation 42.5 0 32.5}
RoadStraight      {translation 42.5 0 37.5}
RoadCross         {translation 42.5 0 42.5}
RoadStraight      {translation 42.5 0 47.5}
RoadStraight2     {translation 42.5 0 52.5}
RoadStraight      {translation 42.5 0 57.5}
RoadCross         {translation 42.5 0 62.5 rotation 0 1 0 3.142}
RoadStraight      {translation 42.5 0 67.5}
RoadStraight2     {translation 42.5 0 72.5}
RoadStraight      {translation 42.5 0 77.5}
RoadT                    {translation 42.5 0 82.5 rotation 0 1 0 3.142}

RoadStraight      {translation 47.5 0  2.5 rotation 0 1 0 1.571}
RoadStraight      {translation 47.5 0 22.5 rotation 0 1 0 1.571}
RoadStraight      {translation 47.5 0 42.5 rotation 0 1 0 1.571}
RoadStraight      {translation 47.5 0 62.5 rotation 0 1 0 1.571}
RoadStraight      {translation 47.5 0 82.5 rotation 0 1 0 1.571}

RoadStraight2     {translation 52.5 0  2.5 rotation 0 1 0 1.571}
RoadStraight2     {translation 52.5 0 22.5 rotation 0 1 0 1.571}
RoadStraight2     {translation 52.5 0 42.5 rotation 0 1 0 1.571}
RoadStraight2     {translation 52.5 0 62.5 rotation 0 1 0 1.571}
RoadStraight2     {translation 52.5 0 82.5 rotation 0 1 0 1.571}

RoadStraight      {translation 57.5 0  2.5 rotation 0 1 0 1.571}
RoadStraight      {translation 57.5 0 22.5 rotation 0 1 0 1.571}
RoadStraight      {translation 57.5 0 42.5 rotation 0 1 0 1.571}
RoadStraight      {translation 57.5 0 62.5 rotation 0 1 0 1.571}
RoadStraight      {translation 57.5 0 82.5 rotation 0 1 0 1.571}

RoadT                    {translation 62.5 0 2.5}
RoadStraight      {translation 62.5 0 7.5}
RoadStraight2     {translation 62.5 0 12.5}
RoadStraight      {translation 62.5 0 17.5}
RoadT                    {translation 62.5 0 22.5 rotation 0 1 0 -1.571}
RoadStraight      {translation 62.5 0 27.5}
RoadStraight2     {translation 62.5 0 32.5}
RoadStraight      {translation 62.5 0 37.5}
RoadT                    {translation 62.5 0 42.5 rotation 0 1 0 -1.571}
RoadStraight      {translation 62.5 0 47.5}
```

```
RoadStraight2     {translation 62.5 0 52.5}
RoadStraight      {translation 62.5 0 57.5}
RoadCross         {translation 62.5 0 62.5 rotation 0 1 0 3.142}
RoadStraight      {translation 62.5 0 67.5}
RoadStraight2     {translation 62.5 0 72.5}
RoadStraight      {translation 62.5 0 77.5}
RoadT                    {translation 62.5 0 82.5 rotation 0 1 0 3.142}

RoadStraight      {translation 67.5 0  2.5 rotation 0 1 0 1.571}
RoadStraight      {translation 67.5 0 62.5 rotation 0 1 0 1.571}
RoadStraight      {translation 67.5 0 82.5 rotation 0 1 0 1.571}

RoadStraight2     {translation 72.5 0  2.5 rotation 0 1 0 1.571}
RoadStraight2     {translation 72.5 0 62.5 rotation 0 1 0 1.571}
RoadStraight2     {translation 72.5 0 82.5 rotation 0 1 0 1.571}

RoadStraight      {translation 77.5 0  2.5 rotation 0 1 0 1.571}
RoadStraight      {translation 77.5 0 62.5 rotation 0 1 0 1.571}
RoadStraight      {translation 77.5 0 82.5 rotation 0 1 0 1.571}

RoadBend          {translation 82.5 0 2.5 rotation 0 1 0 -1.571}
RoadStraight      {translation 82.5 0 7.5}
RoadStraight2     {translation 82.5 0 12.5}
RoadStraight      {translation 82.5 0 17.5}
RoadStraight      {translation 82.5 0 22.5}
RoadStraight      {translation 82.5 0 27.5}
RoadStraight2     {translation 82.5 0 32.5}
RoadStraight      {translation 82.5 0 37.5}
RoadStraight      {translation 82.5 0 42.5}
RoadStraight      {translation 82.5 0 47.5}
RoadStraight2     {translation 82.5 0 52.5}
RoadStraight      {translation 82.5 0 57.5}
RoadT                    {translation 82.5 0 62.5 rotation 0 1 0 -1.571}
RoadStraight      {translation 82.5 0 67.5}
RoadStraight2     {translation 82.5 0 72.5}
RoadStraight      {translation 82.5 0 77.5}
RoadBend          {translation 82.5 0 82.5 rotation 0 1 0 3.142}

# Park Tiles
Park       {translation 12.5 0.07 12.5}
Park       {translation 72.5 0.07 32.5 scale 1 1 3.67}
Park       {translation 12.5 0.07 55 scale 1 1 1.3333}
Park       {translation 22.5 0.07 72.5 scale 2.3333 1 1}

# City Building Tiles
SkyScraper{translation 32.5 0 12.5}
SkyScraper{translation 52.5 0 12.5}
SkyScraper{translation 12.5 0 32.5}
SkyScraper{translation 32.5 0 32.5}
SkyScraper{translation 52.5 0 32.5}
SkyScraper{translation 32.5 0 52.5}
SkyScraper{translation 52.5 0 52.5}
SkyScraper{translation 52.5 0 72.5}
SkyScraper{translation 72.5 0 72.5}
]} # End of world scaling Transform

# Surrounding Terrain
Terrain {}
Windmill {translation  50 9 -10 rotation 0 1 0 -1}
Windmill {translation 100 9 -10 rotation 0 1 0 -1}
Windmill {translation 150 9 -10 rotation 0 1 0 -1}
Windmill {translation 185 14 35 rotation 0 1 0 -1.57}
Windmill {translation 180 7.5 100 rotation 0 1 0 -1}
Windmill {translation 100 8 180 rotation 0 1 0 -1}
Windmill {translation 150 8 180 rotation 0 1 0 -1}
Windmill {translation  50 8 180 rotation 0 1 0 -1}
Windmill {translation -10 9 50 rotation 0 1 0 -1}
Windmill {translation -10 6 150 rotation 0 1 0 -1}

# Checkpoint
DEF checkPt CheckPoint {translation 30 -5 5 scale 2 2 2}

# Player car
DEF carTransform Transform {
    translation 10 0.55 6.5
  children [
    DEF VP_CAR_1 Viewpoint {
```

```
            position -10 2 0
            orientation 0 1 0 -1.57
            description "Behind Car"
    }
    DEF PLAYER_CAR Car {scale 0.3 0.3 0.3 wheelRotation 0 1 0 0 bindCarView TRUE}
  ]
}
DEF VP_CAR_2 Viewpoint {
    position 85 210 85
    orientation -0.577 -0.577 -0.577 2.095
    description "Top Down"
}
ROUTE TS1.isActive TO VP_CAR_2.set_bind

# Traffic
DEF car1 Car {scale .3 .3 .3 carDiffuseColor 0 1 0 carEmissiveColor  0 .4  0
translation 0 -10 0}
DEF car2 Car {scale .3 .3 .3 carDiffuseColor 0 1 1 carEmissiveColor  0 .4 .4
translation 0 -10 0 rotation 0 1 0 3.14}
DEF car3 Car {scale .3 .3 .3 carDiffuseColor 1 1 0 carEmissiveColor .4 .4  0
translation 0 -10 0 rotation 0 1 0 1.57}
DEF car4 Car {scale .3 .3 .3 carDiffuseColor 1 0 1 carEmissiveColor .4  0 .4
translation 0 -10 0 rotation 0 1 0 4.71}
DEF car5 Car {scale .3 .3 .3 carDiffuseColor 0 1 0 carEmissiveColor  0 .4  0
translation 0 -10 0}
DEF car6 Car {scale .3 .3 .3 carDiffuseColor 0 1 1 carEmissiveColor  0 .4 .4
translation 0 -10 0 rotation 0 1 0 3.14}
DEF car7 Car {scale .3 .3 .3 carDiffuseColor 1 0 0 carEmissiveColor .4  0 .0
translation 0 -10 0 rotation 0 1 0 1.57}
DEF car8 Car {scale .3 .3 .3 carDiffuseColor 0 1 0 carEmissiveColor  0 .4  0
translation 0 -10 0}
DEF car9 Car {scale .3 .3 .3 carDiffuseColor 0 1 1 carEmissiveColor  0 .4 .4
translation 0 -10 0 rotation 0 1 0 3.14}

DEF ts1 TimeSensor {
    loop TRUE
    enabled FALSE
    cycleInterval .01
}
DEF ts2 TimeSensor {
    loop TRUE
    enabled FALSE
    cycleInterval 1
}

#counter script
DEF countScript Script {
    eventIn SFTime cycleTime
    eventIn SFBool gameFinished
    eventOut SFBool timerEnabled
    eventOut SFTime startClap
    field SFInt32 clock 5
    field SFNode counttxt USE TEXT1
    field SFNode txtapp USE TXTMAT
    field SFString timeLeft "Time Left:"
    field SFString count ""
    field SFString gameOver "GAME OVER"
    field SFString playerWins "You Have Won!"
    field SFString score "Score:"

    url "javascript:
    function initialize()
    {
    }
    function cycleTime(value,time)
    {
            if(clock > 0) clock--;
            count = new SFString(clock.toString());
            if(clock == 0)
            {
                    counttxt.string = new MFString(gameOver);
                    timerEnabled = false;
            }
            else
            {
                    counttxt.string = new MFString(timeLeft,count);
            }
```

```
            if(clock < 11)
            {
                    txtapp.emissiveColor = new SFColor(1,0,0);
            }
            else
            {
                    txtapp.emissiveColor = new SFColor(1,1,0);
            }
    }
    function gameFinished(value,time)
    {
            if(value==false)
            {
                    counttxt.string = new MFString(playerWins,score,count);
                    startClap = time;
            }

    }
    "
}
ROUTE ts2.cycleTime TO countScript.cycleTime
ROUTE ts2.enabled TO countScript.gameFinished
ROUTE countScript.timerEnabled TO ts1.enabled
ROUTE ts2.enabled TO ts1.enabled
ROUTE countScript.startClap TO AC1.startTime

#Control script
DEF carControl Script {
    eventIn SFInt32 actionKeyPressed
    eventIn SFInt32 actionKeyRelease
    eventIn SFFloat moveCar
    eventOut SFTime soundTime
    field SFFloat wheelAngle 0.0
    field SFFloat posX 10
    field SFFloat posY 0.55
    field SFFloat posZ 6.5
    field SFFloat speed 0
    field SFFloat acceleration .01
    field SFFloat brake 0
    field SFInt32 leftDown 0
    field SFInt32 rightDown 0
    field SFVec3f direction 1 0 0
    field SFNode car USE PLAYER_CAR
    field SFNode cart USE carTransform

    url "javascript:

    function initialize()
    {
            wheelAngle = 0.0;
    }
    function actionKeyPressed(value, time)
    {
            if(value==1006 && leftDown==0)
            {
                    leftDown = 1;
                    print('left!');
            }
            else if(value==1007 && rightDown==0)
            {
                    rightDown = 1;
                    print('right!');
            }
            else if(value==1005)
            {
                    brake = .05;
            }
            soundTime = time;
    }
    function actionKeyRelease(value,time)
    {
            if(value==1006)         leftDown = 0;
            else if(value==1007) rightDown = 0;
            else if(value==1005) brake = 0;

    }
    function moveCar(value,time)
```

```
    {

                    if(speed < .5) speed += (acceleration - brake);
                    if(speed > .5) speed -= brake;
                    if(speed < 0) speed = 0;
                    direction = cart.rotation.multVec(new SFVec3f(1,0,0));
                    posX = speed*direction.x + posX;
                    posY = speed*direction.y + posY;
                    posZ = speed*direction.z + posZ;
                    cart.translation = new SFVec3f(posX,posY,posZ);
    }

    "
    directOutput TRUE
}
ROUTE INPUT_SENSOR.actionKeyPress TO carControl.actionKeyPressed
ROUTE INPUT_SENSOR.actionKeyRelease TO carControl.actionKeyRelease
ROUTE ts1.fraction_changed TO carControl.moveCar
ROUTE carControl.soundTime TO AC2.startTime
```

Appendix B – DLP Source For ccity.pl

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                     %
% Multimedia Authoring II 2006        %
%                                     %
% Marco Bouterse - 1142828            %
%                                     %
% Final Project - "Checkpoint City"   %
%                                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Controller object
%
% Sets up the different opjects

:-object ccity : [bcilib].


% URL of the VRML world
var url = './city/prototypes/city.wrl'.

main :-

    % Enable debug output
    text_area(Browser),
    set_output(Browser),

    % Load the world
    loadURL(url),
    format('The  game will start in 5 seconds,~n'),
     sleep(5000),
    format('The game has started.~n'),


    % Start Traffic
    _Traffic1 := new(traffic(car1, position(41, 0.55, 6.5))),
    sleep(100),

    _Traffic2 := new(traffic(car2, position(129, 0.55, 163.5))),
    sleep(100),

    _Traffic3 := new(traffic(car3, position(86.5, 0.55, 89))),
    sleep(100),

    _Traffic4 := new(traffic(car4, position(3.5, 0.55, 41))),
    sleep(100),

    _Traffic5 := new(traffic(car5, position(41, 0.55, 86.5))),
    sleep(100),

    % Start checkpoints
    _Checkpoint := new(checkpoint),

    % Start player control
    _Car := new(player_car).


:-end_object ccity.
```

```
% Player object
%
% Controls the player's car

:-object player_car : [bcilib].

var distanceBetweenCrossings = 40.

player_car :-
    format('Car thread active.~n'),
    setSFBool(ts1,enabled,true),
    update.


% Car control loop
update :-

    repeat,

            getRotation(carTransform,0,1,0,R),
            getPosition(carTransform,X,_Y,Z),

            nonvar(X),
            nonvar(Z),

            X1 is (X mod distanceBetweenCrossings),
            Z1 is (Z mod distanceBetweenCrossings),

            CX is ((X//distanceBetweenCrossings) + 1),
            CZ is ((Z//distanceBetweenCrossings) + 1),

            decide_action(R,X1,Z1,CX,CZ,Action),

            nonvar(Action),

            do_action(R,Action),

            delay(50),

    fail,
    !.


%decide_action(CarAngle,Xpos,Zpos,CrossingX,CrossingZ,Action)
%
% Finds the action to execute based on current location and player input

% If car is not on a crossing, just go straight
decide_action(_,X,_,_,_,keep_going) :- X > 7, !.
decide_action(_,X,_,_,_,keep_going) :- X < 3, !.
decide_action(_,_,Z,_,_,keep_going) :- Z > 7, !.
decide_action(_,_,Z,_,_,keep_going) :- Z < 3, !.

% If car on a crossing and right key is pressed, turn to right (if possible)
decide_action( 0.0 ,_,_,CX,CZ,turn_right) :-
    getSFInt32(carControl,rightDown,1),
    CZ1 is CZ + 1,
    connected([CX,CZ],[CX,CZ1]),
    !.

decide_action( 1.57,_,_,CX,CZ,turn_right) :-
    getSFInt32(carControl,rightDown,1),
    CX1 is CX + 1,
    connected([CX,CZ],[CX1,CZ]),
    !.

decide_action( 3.14,_,_,CX,CZ,turn_right) :-
    getSFInt32(carControl,rightDown,1),
    CZ1 is CZ - 1,
    connected([CX,CZ],[CX,CZ1]),
    !.

decide_action(-1.57,_,_,CX,CZ,turn_right) :-
    getSFInt32(carControl,rightDown,1),
    CX1 is CX - 1,
    connected([CX,CZ],[CX1,CZ]),
    !.
```

```
% If car is on a crossing and left key is pressed, turn to left (if possible)
decide_action( 0.0 ,_,_,CX,CZ,turn_left) :-
    getSFInt32(carControl,leftDown,1),
    CZ1 is CZ - 1,
    connected([CX,CZ],[CX,CZ1]),
    !.

decide_action( 1.57,_,_,CX,CZ,turn_left) :-
    getSFInt32(carControl,leftDown,1),
    CX1 is CX - 1,
    connected([CX,CZ],[CX1,CZ]),
    !.

decide_action( 3.14,_,_,CX,CZ,turn_left) :-
    getSFInt32(carControl,leftDown,1),
    CZ1 is CZ + 1,
    connected([CX,CZ],[CX,CZ1]),
    !.

decide_action(-1.57,_,_,CX,CZ,turn_left) :-
    getSFInt32(carControl,leftDown,1),
    CX1 is CX + 1,
    connected([CX,CZ],[CX1,CZ]),
    !.

% If there is no input, let car go forward on the crossing if possible
decide_action( 0.0 ,_,_,CX,CZ,keep_going) :-
    CX1 is CX + 1,
    connected([CX,CZ],[CX1,CZ]),
    !.

decide_action( 1.57,_,_,CX,CZ,keep_going) :-
    CZ1 is CZ - 1,
    connected([CX,CZ],[CX,CZ1]),
    !.

decide_action( 3.14,_,_,CX,CZ,keep_going) :-
    CX1 is CX - 1,
    connected([CX,CZ],[CX1,CZ]),
    !.

decide_action(-1.57,_,_,CX,CZ,keep_going) :-
    CZ1 is CZ + 1,
    connected([CX,CZ],[CX,CZ1]),
    !.

% If there is no input and forward is not possible, go right
decide_action( 0.0 ,_,_,CX,CZ,turn_right) :-
    CZ1 is CZ + 1,
    connected([CX,CZ],[CX,CZ1]),
    !.

decide_action( 1.57,_,_,CX,CZ,turn_right) :-
    CX1 is CX + 1,
    connected([CX,CZ],[CX1,CZ]),
    !.

decide_action( 3.14,_,_,CX,CZ,turn_right) :-
    CZ1 is CZ - 1,
    connected([CX,CZ],[CX,CZ1]),
    !.

decide_action(-1.57,_,_,CX,CZ,turn_right) :-
    CX1 is CX - 1,
    connected([CX,CZ],[CX1,CZ]),
    !.

% If there is no input and forward & right are not possible, go left
decide_action( 0.0 ,_,_,CX,CZ,turn_left) :-
    CZ1 is CZ - 1,
    connected([CX,CZ],[CX,CZ1]),
    !.

decide_action( 1.57,_,_,CX,CZ,turn_left) :-
    CX1 is CX - 1,
    connected([CX,CZ],[CX1,CZ]),
```

```
    !.

decide_action( 3.14,_,_,CX,CZ,turn_left) :-
    CZ1 is CZ + 1,
    connected([CX,CZ],[CX,CZ1]),
    !.

decide_action(-1.57,_,_,CX,CZ,turn_left) :-
    CX1 is CX + 1,
    connected([CX,CZ],[CX1,CZ]),
    !.

% If nothing succeeds, return no_action
decide_action(_,_,_,_,_,no_action) :- !.



%do_action(CarRotation,Action)
%
% Executes the given action

% Just let the car keep going forward (handled in vrml)
do_action(_,keep_going) :- !.

% Make the car turn to the right
do_action(-1.57,turn_right) :-
    setRotation(carTransform,0,1,0,3.14),
    delay(1000),
    !.

do_action(R,turn_right) :-
    R1 is R - 1.57,
    setRotation(carTransform,0,1,0,R1),
    delay(1000),
    !.

% Make the car turn to the left
do_action(3.14,turn_left) :-
    setRotation(carTransform,0,1,0,-1.57),
    delay(1000),
    !.

do_action(R,turn_left) :-
    R1 is R + 1.57,
    setRotation(carTransform,0,1,0,R1),
    delay(1000),
    !.

% Not a recognised action
do_action(_,_) :- format('ERROR'), !.


%connected(CrossingA,CrossingB)
%
% Two crossings are connected if there exists a road from A to B or from B to A

connected([X1,Z1],[X2,Z2]) :- road([X1,Z1],[X2,Z2]) ; road([X2,Z2],[X1,Z1]).

%road([X1,Z1],[X2,Z2]
%
% Represents road that connects [X1,Z1] with [X2,Z2]
% The roadmap looks like this:

%
%   1    2    3   4   5
% 1 * - * - * - * - *
%   |   |   |   |   |
% 2 * - * - * - *   *
%   |   |   |   |   |
% 3 * - * - * - *   *
%   |   |   |   |   |
% 4 *   * - * - * - *
%   |       |   |   |
% 5 * - * - * - * - *
%

road([1,1],[2,1]).
```

```
road([1,1],[1,2]).
road([2,1],[3,1]).
road([2,1],[2,2]).
road([3,1],[4,1]).
road([3,1],[3,2]).
road([4,1],[5,1]).
road([4,1],[4,2]).
road([5,1],[5,2]).
road([1,2],[2,2]).
road([1,2],[1,3]).
road([2,2],[3,2]).
road([2,2],[2,3]).
road([3,2],[4,2]).
road([3,2],[3,3]).
road([4,2],[4,3]).
road([5,2],[5,3]).
road([1,3],[2,3]).
road([1,3],[1,4]).
road([2,3],[3,3]).
road([2,3],[2,4]).
road([3,3],[4,3]).
road([3,3],[3,4]).
road([4,3],[4,4]).
road([5,3],[5,4]).
road([1,4],[1,5]).
road([2,4],[3,4]).
road([3,4],[4,4]).
road([3,4],[3,5]).
road([4,4],[5,4]).
road([4,4],[4,5]).
road([5,4],[5,5]).
road([1,5],[2,5]).
road([2,5],[3,5]).
road([3,5],[4,5]).
road([4,5],[5,5]).


:-end_object player_car.



% Checkpoint object
%
% Handles checkpoints and timer


:-object checkpoint : [bcilib].

var nrOfCheckPoints = 20.
var collisionRadius = 3.

checkpoint :-

    format('Checkpoint thread active.~n'),
    setSFBool(ts2,enabled,true),
    game_loop(nrOfCheckPoints).


%game_loop(CheckpointID)
%
% Main control loop for checkpoint thread

game_loop(0) :-

    format('YOU WIN!!'),
    setSFBool(ts2,enabled,false).

game_loop(CP) :-

    checkpoint(CP, X, Y, Z, T),
    setPosition(checkPt,X,Y,Z),

    getSFInt32(countScript,clock,C),
    C1 is C + T,
    setSFInt32(countScript,clock,C1),
```

```
    setSFBool(cpSound,loop,true),
    sleep(800),
    setSFBool(cpSound,loop,false),

    check_collision(X,Z,10),

    CP1 is CP - 1,
    game_loop(CP1).

game_loop(_):- format('ERROR!'), !.


%check_collision(CarX,CarZ,DistanceToCar)
%
% Keeps looping until the distance between the car and checkpoint is small enough

check_collision(_,_,D) :-
    D < collisionRadius,
    setPosition(checkPt,0,-100,0),
    !.

check_collision(X,Z,_) :-
    getPosition(carTransform, Xcar, _, Zcar),
    distance(Xcar,Zcar,X,Z,D1),
    delay(50),
    check_collision(X,Z,D1),
    !.

%distance(X1,Z1,X2,Z2,Distance)
%
% Calculates the distance between two points in 2d plane

distance(X1,Z1,X2,Z2,D) :-

    Xdiff is X2 - X1,
    Zdiff is Z2 - Z1,
    D is (sqrt((Xdiff*Xdiff)+(Zdiff*Zdiff))),
    !.



%checkpoint(id, X, Y, Z, timelimit)
%
% Database of checkpoint locations + time to reach them

checkpoint( 1, 125, 1.5,  65, 10).
checkpoint( 2, 165, 1.5, 125, 10).
checkpoint( 3,  20, 1.5,  45, 10).
checkpoint( 4,  85, 1.5, 145, 10).
checkpoint( 5,  85, 1.5,  65, 10).
checkpoint( 6,  25, 1.5,   5, 10).
checkpoint( 7, 125, 1.5,  25, 10).
checkpoint( 8, 105, 1.5, 125, 20).
checkpoint( 9,   5, 1.5, 125, 20).
checkpoint(10, 165, 1.5,  95, 20).
checkpoint(11,  25, 1.5,  85, 20).
checkpoint(12,  85, 1.5,   5, 20).
checkpoint(13, 125, 1.5, 105, 20).
checkpoint(14,   5, 1.5,  65, 20).
checkpoint(15,  60, 1.5, 165, 30).
checkpoint(16, 165, 1.5, 165, 30).
checkpoint(17,  65, 1.5,  45, 30).
checkpoint(18, 165, 1.5,  25, 30).
checkpoint(19,  45, 1.5, 125, 30).
checkpoint(20,  85, 1.5,  85, 25).


:-end_object checkpoint.
```

```
% Traffic Object
%
% Controls an autonomous car, travelling through the city


:-object traffic : [bcilib].


traffic(Car,position(X,Y,Z)) :-

    format('Traffic thread active [~w].~n',[Car]),
    setPosition(Car,X,Y,Z),

    CX is (X//40) + 1,
    CZ is (Z//40) + 1,

    traffic_loop(Car,CX,CZ).


%traffic_loop(ObjectID,CrossingX,CrossingZ)
%
% Main traffic loop, directing the cars from current location to destination

traffic_loop(Car,CX,CZ) :-

            random(RD1),
            random(RD2),

            CX1 is truncate(RD1*5)+1,
            CZ1 is truncate(RD2*5)+1,

            route([CX,CZ],[CX1,CZ1], Path), !,

            drive_to_destination(Path, Car), !,

            sleep(5000),

            traffic_loop(Car,CX1,CZ1).


%drive_to_destination(Route,Object)
%
% moves object to destination using a calculated route

drive_to_destination([_C1|[]],_) :- !.  % Destination reached if there is only 1
element left

drive_to_destination([C1,C2|Path2],Car) :-

    getRotation(Car,0,1,0,CarRotation),
    decide_action(CarRotation,C1,C2,Action),
    nonvar(Action),

    getPosition(Car,X,Y,Z),
    do_action(Car,CarRotation,position(X,Y,Z),Action),

    Path3 = [C2|Path2],
    drive_to_destination(Path3,Car).

drive_to_destination(_,_) :- format('ERROR~n'), !.


%decide_action(CarRotation,CurrentCrossing,DestinationCrossing,Action)
%
% Decides which action must be taken to reach the destination crossing

decide_action(0.0, [CX1,CZ], [CX2,CZ], go_straight) :- CX1 < CX2, !.
decide_action(1.57,[CX,CZ1], [CX,CZ2], go_straight) :- CZ1 > CZ2, !.
decide_action(3.14,[CX1,CZ], [CX2,CZ], go_straight) :- CX1 > CX2, !.
decide_action(4.71,[CX,CZ1], [CX,CZ2], go_straight) :- CZ1 < CZ2, !.

decide_action(0.0, [CX,CZ1], [CX,CZ2], turn_right) :- CZ1 < CZ2, !.
decide_action(1.57,[CX1,CZ], [CX2,CZ], turn_right) :- CX1 < CX2, !.
decide_action(3.14,[CX,CZ1], [CX,CZ2], turn_right) :- CZ1 > CZ2, !.
decide_action(4.71,[CX1,CZ], [CX2,CZ], turn_right) :- CX1 > CX2, !.

decide_action(0.0, [CX,CZ1], [CX,CZ2], turn_left) :- CZ1 > CZ2, !.
```

```
decide_action(1.57,[CX1,CZ], [CX2,CZ], turn_left) :- CX1 > CX2, !.
decide_action(3.14,[CX,CZ1], [CX,CZ2], turn_left) :- CZ1 < CZ2, !.
decide_action(4.71,[CX1,CZ], [CX2,CZ], turn_left) :- CX1 < CX2, !.

decide_action(0.0, [CX1,CZ], [CX2,CZ], turn_back) :- CX1 > CX2, !.
decide_action(1.57,[CX,CZ1], [CX,CZ2], turn_back) :- CZ1 < CZ2, !.
decide_action(3.14,[CX1,CZ], [CX2,CZ], turn_back) :- CX1 < CX2, !.
decide_action(4.71,[CX,CZ1], [CX,CZ2], turn_back) :- CZ1 > CZ2, !.

decide_action(_,_,_,no_action) :- !.


%do_action(Object,Orientation,Position,Action)
%
% Moves the object from current crossing to the next one

do_action(Car,0.0 ,position(X,Y,Z),go_straight) :-
    X1 is X + 40,
    move_to_position(Car,position(X,Y,Z),position(X1,Y,Z),40),
    !.

do_action(Car,1.57,position(X,Y,Z),go_straight) :-
    Z1 is Z - 40,
    move_to_position(Car,position(X,Y,Z),position(X,Y,Z1),40),
    !.

do_action(Car,3.14,position(X,Y,Z),go_straight) :-
    X1 is X - 40,
    move_to_position(Car,position(X,Y,Z),position(X1,Y,Z),40),
    !.

do_action(Car,4.71,position(X,Y,Z),go_straight) :-
    Z1 is Z + 40,
    move_to_position(Car,position(X,Y,Z),position(X,Y,Z1),40),
    !.


do_action(Car,0.0 ,position(X,Y,Z),turn_right) :-
    setRotation(Car,0,1,0,-0.785),
    X1 is X + 2.5, Z1 is Z + 2.5,
    move_to_position(Car,position(X,Y,Z),position(X1,Y,Z1),5),
    setRotation(Car,0,1,0,4.71),
    Z2 is Z1 + 32,
    move_to_position(Car,position(X1,Y,Z1),position(X1,Y,Z2),32), !.

do_action(Car,1.57,position(X,Y,Z),turn_right) :-
    setRotation(Car,0,1,0,0.785),
    X1 is X + 2.5, Z1 is Z - 2.5,
    move_to_position(Car,position(X,Y,Z),position(X1,Y,Z1),5),
    setRotation(Car,0,1,0,0.0),
    X2 is X1 + 32,
    move_to_position(Car,position(X1,Y,Z1),position(X2,Y,Z1),32), !.

do_action(Car,3.14,position(X,Y,Z),turn_right) :-
    setRotation(Car,0,1,0,2.355),
    X1 is X - 2.5, Z1 is Z - 2.5,
    move_to_position(Car,position(X,Y,Z),position(X1,Y,Z1),5),
    setRotation(Car,0,1,0,1.57),
    Z2 is Z1 - 32,
    move_to_position(Car,position(X1,Y,Z1),position(X1,Y,Z2),32), !.

do_action(Car,4.71,position(X,Y,Z),turn_right) :-
    setRotation(Car,0,1,0,3.925),
    X1 is X - 2.5, Z1 is Z + 2.5,
    move_to_position(Car,position(X,Y,Z),position(X1,Y,Z1),5),
    setRotation(Car,0,1,0,3.14),
    X2 is X1 - 32,
    move_to_position(Car,position(X1,Y,Z1),position(X2,Y,Z1),32), !.


do_action(Car,0.0 ,position(X,Y,Z),turn_left) :-
    setRotation(Car,0,1,0,0.785),
    X1 is X + 5.5, Z1 is Z - 5.5,
move_to_position(Car,position(X,Y,Z),position(X1,Y,Z1),10),
    setRotation(Car,0,1,0,1.57),
    Z2 is Z1 - 32,
    move_to_position(Car,position(X1,Y,Z1),position(X1,Y,Z2),32), !.
```

```
do_action(Car,1.57,position(X,Y,Z),turn_left) :-
    setRotation(Car,0,1,0,2.355),
    X1 is X - 5.5, Z1 is Z - 5.5,
move_to_position(Car,position(X,Y,Z),position(X1,Y,Z1),10),
    setRotation(Car,0,1,0,3.14),
    X2 is X1 - 32,
    move_to_position(Car,position(X1,Y,Z1),position(X2,Y,Z1),32), !.

do_action(Car,3.14,position(X,Y,Z),turn_left) :-
    setRotation(Car,0,1,0,3.925),
    X1 is X - 5.5, Z1 is Z + 5.5,
move_to_position(Car,position(X,Y,Z),position(X1,Y,Z1),10),
    setRotation(Car,0,1,0,4.71),
    Z2 is Z1 + 32,
    move_to_position(Car,position(X1,Y,Z1),position(X1,Y,Z2),32), !.

do_action(Car,4.71,position(X,Y,Z),turn_left) :-
    setRotation(Car,0,1,0,-0.785),
    X1 is X + 5.5, Z1 is Z + 5.5,
move_to_position(Car,position(X,Y,Z),position(X1,Y,Z1),10),
    setRotation(Car,0,1,0,0.0),
    X2 is X1 + 32,
    move_to_position(Car,position(X1,Y,Z1),position(X2,Y,Z1),32), !.


do_action(Car,0.0 ,position(X,Y,Z),turn_back) :-
    setRotation(Car,0,1,0,1.57),
    Z1 is Z - 3, move_to_position(Car,position(X,Y,Z),position(X,Y,Z1),3),
    setRotation(Car,0,1,0,3.14),
    X1 is X - 32,
    move_to_position(Car,position(X,Y,Z1),position(X1,Y,Z1),32), !.

do_action(Car,1.57,position(X,Y,Z),turn_back) :-
    setRotation(Car,0,1,0,3.14),
    X1 is X - 3, move_to_position(Car,position(X,Y,Z),position(X1,Y,Z),3),
    setRotation(Car,0,1,0,4.71),
    Z1 is Z + 32,
    move_to_position(Car,position(X1,Y,Z),position(X1,Y,Z1),32), !.

do_action(Car,3.14,position(X,Y,Z),turn_back) :-
    setRotation(Car,0,1,0,4.71),
    Z1 is Z + 3, move_to_position(Car,position(X,Y,Z),position(X,Y,Z1),3),
    setRotation(Car,0,1,0,0.0),
    X1 is X + 32,
    move_to_position(Car,position(X,Y,Z1),position(X1,Y,Z1),32), !.

do_action(Car,4.71,position(X,Y,Z),turn_back) :-
    setRotation(Car,0,1,0,0.0),
    X1 is X + 3, move_to_position(Car,position(X,Y,Z),position(X1,Y,Z),3),
    setRotation(Car,0,1,0,1.57),
    Z1 is Z - 32,
    move_to_position(Car,position(X1,Y,Z),position(X1,Y,Z1),32), !.


do_action(_,_,_,_) :- format('Unrecognised Action~n'), !.


%move_to_position(Object,CurrentPosition,DestinationPosition,InterpolationFactor)
%
% Interpolates an object's position between two points

move_to_position(Object,_,position(X,Y,Z),0):-
    setPosition(Object,X,Y,Z).

move_to_position(Object, position(X1,Y1,Z1), position(X2,Y2,Z2),C):-

    C1 is C-1,

    Xdif is X2 - X1,
    Zdif is Z2 - Z1,

    X is X1 + Xdif/C,
    Z is Z1 + Zdif/C,

    setPosition(Object,X,Y1,Z),
    sleep(50),
```

```
    move_to_position(Object,position(X,Y1,Z), position(X2,Y2,Z2),C1).


%route(CrossingA,CrossingB,RouteFromAtoB)
%
% Calculates route from A to B

route(A, B, Path) :- route_hlp(A, B, Path, [A]).

route_hlp(A, B, Path,_) :-
    connected(A,B),
    Path = [A,B].

route_hlp(A, B, Path, Between) :-
    connected(A,C),
    \+ member(C, Between),
    Between2 = [C|Between],
    route_hlp(C, B, Path2, Between2),
    Path = [A| Path2].


%connected(CrossingA,CrossingB)
%
% Two crossings are connected if there exists a road from A to B or from B to A

connected([X1,Z1],[X2,Z2]) :- road([X1,Z1],[X2,Z2]) ; road([X2,Z2],[X1,Z1]).


%road([X1,Z1],[X2,Z2]
%
% Represents connected roads

road([1,1],[2,1]).
road([1,1],[1,2]).
road([2,1],[3,1]).
road([2,1],[2,2]).
road([3,1],[4,1]).
road([3,1],[3,2]).
road([4,1],[5,1]).
road([4,1],[4,2]).
road([5,1],[5,2]).
road([1,2],[2,2]).
road([1,2],[1,3]).
road([2,2],[3,2]).
road([2,2],[2,3]).
road([3,2],[4,2]).
road([3,2],[3,3]).
road([4,2],[4,3]).
road([5,2],[5,3]).
road([1,3],[2,3]).
road([1,3],[1,4]).
road([2,3],[3,3]).
road([2,3],[2,4]).
road([3,3],[4,3]).
road([3,3],[3,4]).
road([4,3],[4,4]).
road([5,3],[5,4]).
road([1,4],[1,5]).
road([2,4],[3,4]).
road([3,4],[4,4]).
road([3,4],[3,5]).
road([4,4],[5,4]).
road([4,4],[4,5]).
road([5,4],[5,5]).
road([1,5],[2,5]).
road([2,5],[3,5]).
road([3,5],[4,5]).
road([4,5],[5,5]).

:-end_object traffic.
```