BIL 421/BIL 539:Spring 2020

Assignment 1

Assigned Thursday, Januray 16. The program must be submitted to our grader (sksueksioglu@gmail.com) by Thursday, January 30 (any time up to midnight). Subject of the email should say "Windows: BIL421 Assignment 1". Replace "Windows" with "Linux" or "MacOS" if your code is to be compiled in those operating systems. Name your Project directory as "YournameAssg1". Before submitting, **zip** your Project directory, and email a single zipped file as attachment. Do NOT include any .exe or .o files in your submission. Remember that your code should be fully documented. Check the course syllabus for the late policy. I also would like to remind you once again about the academic honesty rules stated in the syllabus. The implementation will be done using immediate mode (pre 3.0) OpenGL, in other words NOT shader-based.

Overview: In this assignment you will implement a 2-dimensional video game as described below. You are allowed to implement extra features, as long as you provide sufficient documentation in your README file of how to play the game (You have to provide a README file with your submission even if you do not add any extra features.)

The game takes place on a screen that is 600 high by 500 wide. Your game scene will consist of 5 roads parallel to the x-axis (horizontal roads) and 6 sidewalks (4 separating the roads, 1 at the bottom and 1 at the top of the screen). Each road has 3 or 4 lanes, mark the lanes with broken lines. See the figure below. (As longs as you have a similar layout, you are free to increase the number of roads/lanes and your window size.)

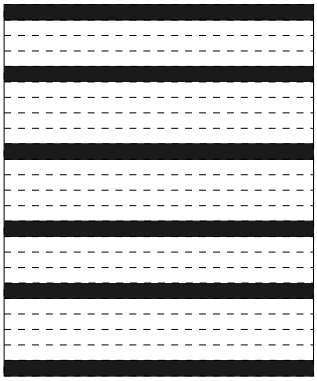

Figure 1: Layout

Game Objects: There are a number of game objects which may be moving during the course of the game. Here are their specifications:

The vehicles (cars and trucks): There will be cars and trucks moving within the lanes. You may decide on the direction of travel for each lane initially. A car is drawn as a square, and a truck is drawn as a rectangle whose width is at least twice as its height. Make the cars and

trucks heights slightly less than the lane width so they are always within lane boundaries. Your program will randomly create cars and trucks in random lanes, and they will be moving with some constant velocity in their lane. You may decide on what their velocity is as you create them. These vehicles do NOT change their initial lanes. When created, a vehicle will appear as if it is entering the scene from the left or from the right (also chosen randomly), and as they reach the other end they disappear from the scene. You need a dynamic data structure to store the cars and trucks which are visible. Create them when they enter the scene and delete them from your data structure as they no longer are visible.

The agent: The user will control an *agent* that initially sits at the bottom sidewalk exactly in the middle. The agent will be drawn as an isosceles triangle whose two equal length edges are longer than the third edge. Let's call the vertex opposite to the shorter edge the tip of the agent. The goal of the agent is to keep crossing roads without being hit by a car or truck. The agent has a *c*urrent moving direction which will be initially UP. The tip of the agent should be pointing in its current direction, therefore initially the tip is pointing up. While the moving direction is UP, if the user presses the up arrow, the agent jumps one step forward. A *step* is defined as a movement from the center of one lane to the center of the next lane, from a sidewalk to the lane following it or from a lane to the sidewalk. If you make your sidewalks wider than one step, it is also possible that the agent can take multiple steps on the sidewalk. When the agent reaches the top sidewalk its direction will be updated to DOWN and the tip of the triangle should be pointing down and the agent can now jump one step downward when the user presses the down key. If the agent now reaches the bottom of the screen, it turns upward again and the game continues indefinitely like this until the agent is hit by a vehicle. If you wish, you can also limit the duration of the game by keeping time, or you can terminate the game if the user reaches a certain score and declare the user a winner.

You will be keeping a score for the user which should be printed somewhere reasonable on your screen. For each jump in the current direction the user earns 1 points. The user can move the agent left and right by pressing left and right arrow keys. (No need to change the tip direction as the agent moves left or right, keep it in its current orientation). If the agent is hit by a vehicle, the game is over. In addition, if the user tries to move in the opposite of its current direction, the game is over.

The coins: Occasionally, some coins may show up on a lane and disappear after a short period. Your program will also generate these coins randomly (but less frequently than cars and trucks). The coins should be drawn as yellow *filled circles*. If the agent moves over a coin, it collects the coin, it is an additional 5 points. The coin disappears if collected.

The actual constant factors are up to you, depending on what looks good on your system.

Other Requirements:

- Left button down: When the left mouse button is pressed, the game pauses. When the left button is pressed a second time the game resumes.
- Right button down: When the right button is pressed, the game pauses if it is in a running state. (If it is already in a paused state or if the game is over, it stays in its current state.)

The state of the game is advanced by one step, and then the program outputs all the game's current state to the standard output (for debugging purposes). This should include the locations of the agent, the vehicles, and any other information that you find important. By repeatedly hitting the right button the program will perform consecutive single steps.

- Keyboard: When the 'q' key is hit, the program quits.

Implementation Suggestions:

The game simulation can be advanced by one step for each timer event. Each moving game object (agent, car, truck) is associated with a location and a velocity (speed and direction) and the object is moved by this velocity at each step of the game. If a car or truck leaves the screen area, then it ceases to exist. If the agent reaches the top of the screen, it turns facing downward and keeps moving downward until reaching the bottom of the screen at which time it turns upward again and the game continues like this.

To find out whether a car or truck hits the agent, in general, you can test whether the bounding box of vehicle intersects with the bounding box of the agent (you can use a rectangular bounding box). Or, you can keep track which lane the agent is on, and if there is a car/truck at the agent coordinates in that same lane you can consider it a hit. This is because the agent is jumping from one lane to another in each step. Also to find out if agent was able to collect a coin, you can check if the square bounding box of the coin intersects the bounding box of the agent. This is approximately correct but is okay. When the game is over, the image pauses and the program waits for the user to hit the 'q' key to terminate the program.

Grading Policy: Grading will be based on the following points. Try to implement and test one feature fully before proceeding with the next feature, so that if you do not complete the project you still get partial credit.

- Correct layout, correct agent display and movement: 25%.
- Correct car and truck display and movement: 25%
- Correct collision detection: 20%
- Correct coin appearance/disappearance and coin collection 10%
- Pause and single step: 10%
- Documentation and clarity: 10%

Extra Credit Features: Feel free to add any other features for extra credit. Explain them in your Readme file.

START EARLY!