SEN224 – System Programming Lab

Spring 2024 –Project Assignment

Due: Sunday, 09/06/2024, 23:59

Requirements:

1. Only properly demonstrated assignments will be graded.
2. Assignments should be solved individually, cheating will be penalized. Do not share your answers with other students.
3. Submit the assignment by the deadline, electronically through Google Classroom.
4. Late submissions will not be accepted.
5. Just submit your .c and .h files.
6. Archive your files into a .zip file. Use your student number to build your filename. Example: "21*******.zip"

CAR TRAFFIC GAME

In this project, you are going to write a car traffic game for children. Briefly, the game you are asked to write is as follows: There should be cars moving from top to bottom in a one-way traffic. At the end of the road there must be a car that the player can control. The game must continue as long as the car the player is driving does not collide with any car in traffic. The player will earn points for each cars player avoids colliding with.

When the program first runs, the main menu should appear and all sections here should be written as specified in this file.

Figure 1 Main menu

1. GAME REQUIREMENTS

The basic functions that should be in the game are:

- The game must take place on a central road of 100 height and 40 width.
- Traffic on the road must be one-way and flow only from top to bottom.
- There should be a line in the middle of the road that divides the road into two, and cars should either drive on the left or right of this line.
- The size (height and width), color, car pattern (*, +, #) and speed of each car should be different.
- Each car should be able to move independently of each other at different speeds.
- There should be a car that the player can move (left or right only) at the end of the road.
- New cars should enter traffic at different intervals.

- If the car the player is driving collides with another car in traffic, the game must end.
- The player must earn points for each car player avoids colliding with. Each car should have different points (smaller car less points, larger car more points).
- As the game progresses, cars must move faster and faster.
- Cars that reach the end of the road should disappear from the screen.
- The game played should be able to be saved and the saved game should be replayable later.
- The player should be able to move the car with different key combinations. For this, the player should be given a choice.
- The player should be able to view the points player earned for each game later.
- The player should be able to view the game instructions.

2. GAME DESIGN

- The game must take place on a road with a height of 100 and a width of 40.
- There should be a line in the middle of the road dividing the road into two. The x coordinate of this line is 45. The distance of the line from both the beginning and the end of the road is 10.
- Each car in the game is implemented as a struct. The UML diagram of this struct is given in Figure 2:

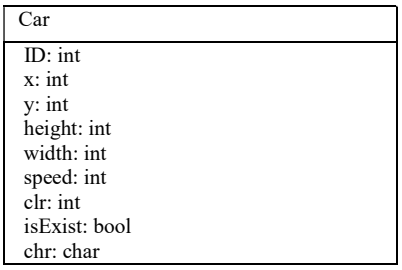


Figure 2 UML diagram of the Car struct

  - **ID:** uniquely identifies the car. The initial value is 10.
  - **x:** x coordinate of the car on the screen. It is determined randomly and ranges from 5 to 90.
  - **y:** y coordinate of the car on the screen. It is randomly determined and ranges from -10 to 0.
  - **height:** length of the car. It is determined randomly and is between 5 and 7.
  - **width:** the width of the car. It is determined randomly and is between 5 and 7.
  - **speed:** the speed of the car. It is half the height of the car.
  - **clr:** color of the car. It is determined randomly and is between 1 and 4.
  - **isExist:** indicates whether the car has exited the screen, initial value is false.
  - **chr:** the character used when the car is drawn on the screen. It is determined randomly. It takes one of the values '*', '#' or '+'.

2.1 New Game

Control parameters for each game are hold in a struct. The UML diagram for this structure is given in Figure 3.

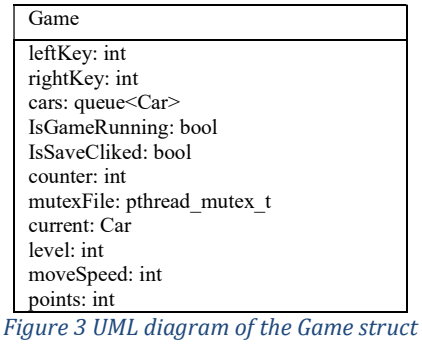


Figure 3 UML diagram of the Game struct

- **leftKey:** holds the ASCII code of the key that the player will use as the left key. **The default value is 260 (left arrow key). This value is assigned when the program is first opened, not when a new game starts.**
- **rightKey:** holds the ASCII code of the key that the player will use as the right key. **The default value is 261 (right arrow key). This value is assigned when the program is first opened, not when a new game starts.**
- **cars:** New cars that will be put on the road are first put into this queue. A maximum of 5 cars can queue. When the game first starts, the queue starts empty.
- **IsGameRunning:** Indicates whether the game is running or not. The initial value is true.
- **IsSaveCliked:** Indicates whether the player pressed the "Save" button (S). The initial value is false.
- **counter:** keeps information on how many cars are generated. The initial value is 10. This value increases as the new car is generated. **When a new car is produced, it is assigned to the car's ID. If it exceeds 20, it is set back to 10.**
- **mutexFile:** mutex variable to be used when writing car information to the file while saving the game.
- **current:** Keeps the information of the car that the player will control. The initial values for this car are given in the template code.
- **level:** Shows the current level of the game. This level increases as the player passes certain score thresholds. The initial value is 1. The maximum level is 5.
- **moveSpeed:** controls how many microseconds the cars in traffic will be moved. The starting value is 500000.
- **points:** keeps the points earned by the player. The initial value is 0.

- **For each new game, the control parameters take initial values and a new game begins.**
- While each car is drawn on the screen, the x and y coordinates of the car are navigated and a rectangle with the height and width of the car is drawn. Then, the car's point is written at the top left of this rectangle. This point is calculated as: height of the car * width of the car. Finally, if this car is a car controlled by the player, another line is drawn on the top of the rectangle, otherwise one more line is drawn on the bottom.
- Current point information is printed in the lower right corner of the screen. The coordinate information to be printed is as follows: x = 91, y = 42.
- On the right side of the road, 3 trees should be drawn in Figure 4.
- The distance of the first tree to the road is 5 pixels and its y coordinate is 5.
- The distance between trees is 10 pixels.

An example start screen of the game is given in Figure 4.

Figure 4 An example start screen of the game

- During the game, new cars to be added to the traffic are generated and added to the cars queue first. This process is repeated every 1 second. If the queue is full, no new car is added to the queue. **We call this process the EnQueue process.**
- **No car should be produced to pass over the line in the middle of the road.**

- Every 2 or 4 seconds, a car is picked up from the queue and starts to move out into traffic. **We call these operations DeQueue process and MoveCar process, respectively.**
- **The MoveCar process is performed independently for each car.**
- Each car in traffic is moved every microsecond according to moveSpeed of the game. The movement of the car is carried out as follows:
    - y of the car += 1 + rand() % speed of the car

An example image of cars in motion is given in Figure 5.

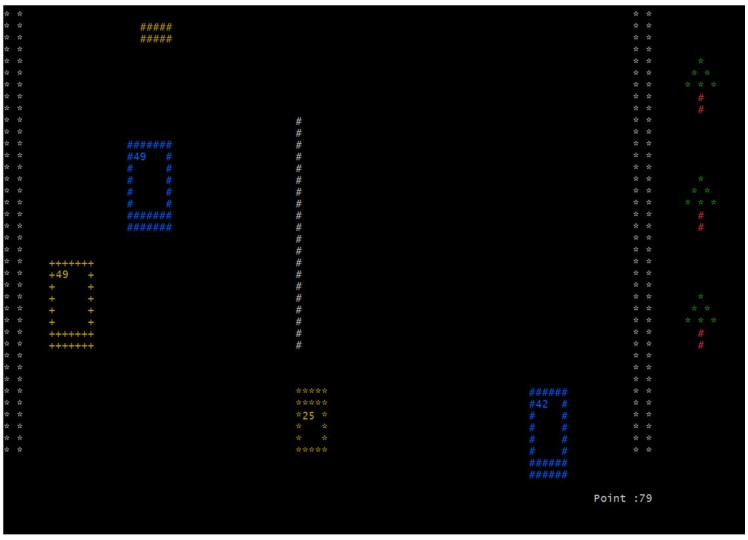


Figure 5 An example image of cars in motion

- After each move during the MoveCar process, it is checked whether the car leaves the screen or whether there is a collision.
- When the cars reach the end of the road, they disappear from the screen. If the car's y coordinate value exceeds 35, it indicates that it has reached the end of the road.
- The player earns point for each car that leaves the screen. At this point, the game point is updated (current point + car's point) and the current point is printed on the screen.
- After every 300 points earned, if the game level has not reached 5, the game level is increased by 1 and moveSpeed of the game is reduced by 100000.
- If the player presses the left button, the car the player controls moves to the left (x-=speed), and if the player presses the right button, it moves to the right (x+=speed). However, the car must not go off the road. While the x coordinate must be greater than 2 for it to move left, the x coordinate must be less than 93 for it to move right. **While waiting for the player to press the arrow keys, 250000 microseconds must be waited each time.**
- If the car the player controls collides with any car in the traffic, the game ends. At the end of each game, the points earned are written to the "points.txt" file.
- If the player presses the ESC button, the game ends and returns to the main menu.
- If the player presses the Save button: (1) The game information is written to the "game.txt" file. (2) The information of the cars in traffic are written to the "cars.txt" file. Writing is not performed for cars in the queue. (3) The game ends and returns to the main menu.

2.2 Load the last game

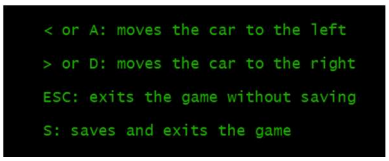
- If there is a saved game, game information is read from the game.txt file. Then game is started with the information read.
- Immediately afterwards, the information of the cars saved in the cars.txt file should be read and the moveCar process should be started for each of them.
- There must be a wait of 250000 microseconds between each reading operation.
- If there is no saved game, a new game must be started with initial values.

Apply these rules for screen printing in Sections 2.3, 2.4, and 2.5:

- The starting coordinate of the first row should be x = 10, y = 5.
- Then there should be a distance of 2 pixels between each line.
- If the number of rows exceeds 10, go back to the beginning and move on to the second column.
- There should be 20 pixels between each column.
- If the user is asked to select options with arrow directions, 200000 microseconds must be waited before each input.

2.3 Instructions

- What the keys used during the game do should be written on the screen.
- After 5 seconds, you should return to the main menu.

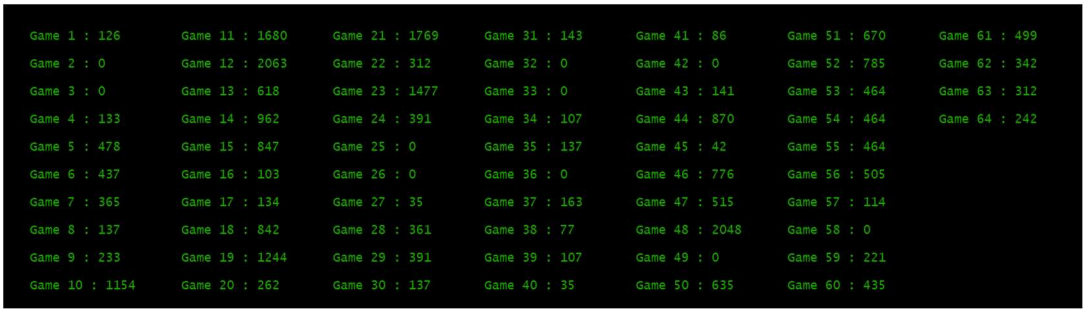


2.4 Settings

- The player should be able to choose which keys player wants to play with. Two options should be offered for this (1) Play with < and > arrow keys (2) Play with A and D keys.
- If the player chooses the first option, the left key should be set as the left arrow key and the right key should be set as the right arrow key in the game settings.
- If the player chooses the second option, the left key should be set to A and the right key should be set to D in the game settings.
- Finally, you should return to the main menu.

2.5 Points

- Read the points in the Points.txt file and print them to the screen.
- After 5 seconds, you should return to the main menu.



3. GAME IMPLEMENTATION DETAILS

- **Each section must be performed in a new window.**
- When a new game starts, a thread that will manage the new game should be run and this thread should be waited until the game is finished.
- Then, separate threads must be run for EnQueue and DeQueue operations. In this way, both jobs can be done at the same time.
- During the DeQueue process, the moveCar process must be started for each car and this process must be executed by a new thread. In this way, each car can move independently of each other.

- The game.txt and cars.txt files must be used in binary form. For this, fread and fwrite functions should be used.
- The points.txt file is not in binary structure. Writing to this file must be done using fprintf/fscanf functions.
- Mutex must be used when writing to the Cars.txt file.

ADDITIONAL INFORMATION

- We share two files with you: game.exe and game.cpp.
- The game.exe is the working versions of the requested game. Try running the game.exe file on cygwin.
- This assignment will be done in a group of two students.
- You must send an e-mail to oayana@atu.edu.tr with information about who you will be in a group with by April 24, 23.59.
- I will randomly match students who do not provide group information.
- **To compile game.cpp you should use this code: g++ game.cpp -o game -lncurses**

ASSIGNMENT SUBMISSION

- The game.cpp file is template code of the project. Just fill in the game.cpp and submit.
- Information about who and how wrote the functions must be written above the function as a comment line. Care should be taken to write common functions equally.
- Students within the group must submit their solutions individually.

IMPLEMENTATION CONSTRAINTS:

- Standard I/O functions must be used for I/O operations (eg: fread, fwrite, fprintf, fscanf, fseek).
- The game.txt and cars.txt files must be used in binary form.
- The points.txt file must be used in the normal form.
- **Defining a new global variable is prohibited. New parameters cannot be added to the game.**