Car Racing Game

Raymond Chan
UNIVERSITY OF CALIFORNIA, DAVIS
Department of Electrical and Computer Engineering
EEC 172 Embedded System Design Spring 2020
Davis, CA, United State
rwfchan@ucdavis.edu

***Abstract*—The CC3200 LaunchPad's accelerometer and Inter-Integrated Circuit (I^2C) offers interesting interaction with the Adafruit OLED Breakout Board. The Breakout Board can react to the LaunchPad's accelerometer at a reasonable speed. However, the pixel update speed on the Board could limit what the application can achieve. This paper is to introduce a car racing game in which its game mechanic utilizes the interactions mentioned above. In the end, the game's implementation can respond to LaunchPad's accelerometer in real-time, while updating the player and other game objects, as well as the scoring and speed display in a relatively smooth manner.**

I. Introduction

Car racing games had always been some variants of the player controlling the main car object in either first or third person, which is moving on a platform that resembles some sort of drive road. The main objective of the game is usually to get first place of the race if other competing, possibly non-player control car objects are present. Other times, the game could be about avoiding obstacles or colliding with as many obstacles as possible. Typically, the game might incentify players to move towards the goal or provide players extra challenges by keeping track of some sort of score, where the game will reward the player so when the player successfully performs a particular task.

The car racing game this paper discusses wants its player to avoid as many obstacles as possible, where each obstacle is non-player controlled car objects. Therefore, the goal of the player is to survive as long as possible.

As of today, there were and are countless car racing games created, ran, and hosted by a different set of software, hardware, and server. Depending on the components and programs used to implement the game, the game appearance and mechanic can have dramatic differences and limitations. For this car racing game, it is implemented using the CC3200 LaunchPad for control and Adafruit OLED Breakout Board for display.

II. Motivation

There are two main motivation for using LaunchPad's accelerometer and the OLED Breakout Board to implement a car racing game: (1) to decide a game mechanic that incorporate LaunchPad's sensitive accelerometer; (2) to understand the limitations of the speed of the Breakout Board display and how it would limit the game's mechanic and performance.

A. Incorporating LaunchPad's Accelerometer

For a typical video game player, they expect an entertaining game to behave reasonably, predictably, while still maintain a few uncertainly or skill-related challenges to constantly capture the player's attention. Although the LaunchPad and the OLED Breakout Board offer interesting functionality and allow programmers to create fascinating creation, the LaunchPad's sensory is basic (and can at times be unreliable), and the Breakout Board is slow at displaying or changing pixel in a fast pace. When exploring LaunchPad's accelerometer, one might discover that the accelerometer has a high sensitivity to the pad's motion. In particular applications, this high sensitivity might not be desirable when the user of the application expects a full extend of control to the accelerometer controlled object. In this case, the player of the game (users) expects the player they control (an object that responds to the accelerometer's data stream) to behave exactly how they wanted it to behave. Furthermore, each of the game states should be predictable yet challenging, so that player can stay engaging while thinking the game is still being fair.

B. Understanding the Limit of OLED Breakout Board Pixel Update

The problem with Breakout Board pixel update is that while the board is slowly changing the presentation of indicated screen positions, the game itself is continuously running lines of codes. When integrating game mechanic into pixel updating, one needs to beware that updating graphic on the board does not cause visual distortion due to it overlapping with other ongoing graphic updates, and graphic representation is on par with how the game reacts to the player's action (for example, if the graphic indicate the non-player control car object to be at particular x and y position, the player car object should be able to interact with it when the player car object's graphic overlaps with that non-player control car object's graphic). One of the challenge when working with the Breakout Board is to implement game mechanics with the limiting speed of pixel update in mind.

III. Game Mechanic and Implementation

As mentioned above, the objective of the game is to have the player-controlled car object survive as long as possible without colliding with any other non-player car objects, while getting the highest score possible. The player will play this game in

a top-down, birds-eye view that looks down onto a 3 lanes highway. The player car object's speed is being kept track of by a global speed counter. On each lane, the player can move up or down, which changes the global speed of the player car object accordingly. Players can also choose to jump from each of their current lanes onto the adjacent lanes, although this decision will not change the global speed. Meanwhile, the change of global speed will change the non-player car object's appearance speed based on their base speed. Non-player car objects' spawn location on each of their corresponding lanes is also depending on the current global speed, forcing players to dodges the non-player car objects via lane switching and adjusting their global speed (all of which will discuss in later subsections). Both the player and the non-player car objects only update once per set amount of time frame, which is keeping track of by a global timer counter. The global timer counter only increments by one per every complete update routine (player car object movement update, non-player car object movement update, and non-player car object spawning). In this paper, this timer counter incrementation will be referred to as *a tick* per one incrementation. The player's score is also keeping track of and will increase or decrease depending on the player's actions.

A. Player Control

At any instant, the player-controlled car object occupies one of the three-car lanes. Players can change their lanes by using LaunchPad's accelerometer. When the player tilts the pad to the left (with respect to the USB connection facing away from the user), the player-controlled car object will jump to its adjacent left car lane instantly, and the same with when the player tilts the pad to the right. If the current lane the player car object is on is the left/rightmost lane, tilt the pad to the left/right, respectively, will not move the car object at all.

Again, the challenge with controlling the player car object with the accelerometer is that the accelerometer could be overreacting to player's tilting motion, so when the player wanted their car object to only move one lane to the left or right, the accelerometer might jump two lanes instead. Meanwhile, the players should be allowed to jump two lanes if they choose to do so. One of the solutions to this problem is by adding delay to how often the player car object can be updated. The delay cannot be too long, as that will make players feel the lack of control over their car object, thus diminishing their gaming experience. The delay cannot be too short either, otherwise, there would be no point to add the delay in the first place. The delay of 5 ticks is chosen for the player car object's update interval, i.e. the player car object will only update every 5 ticks relative to the global timer.

Players can also move up and down the lane by pressing the SW3(left) button on the LaunchPad for up and SW2(right) button for down, and it cannot go off-screen. When the player car object is moving up, the game will also consider the player car object is accelerating, therefore the global speed counter will increase as the player keep pressing onto the SW3 button. On the other hand, the SW2 button is considered as braking, meaning while the player car object is moving down visually on its lane, its speed is also decelerating. The player's initial global speed is 50 units. The global speed can in theory be as large as `MAX_INT` and as small as -`MAX_INT`. In practice, the game will become too difficult if the magnitude of the global speed becomes too large (will discuss in subsection B. Non-Player Car Object Movement).

B. Non-Player Car Object Movement

The non-player car object's movement is being updated every 20 ticks (The reason for why 20 ticks are chosen instead of unifying its update ticks with the player car object's update ticks will be discussed in section IV. Game Mechanic and Graphic Representation). Each non-player car object will only move up or down on its lane but neither switch lanes. The non-player car objects have different moving base speed depending on which lane it is in. More specifically, the base speed for the left-most lane (LANE1) is 90 units, the middle lane is 40 units, and the rightmost lane is 70 units. The base speed is not the same as the global speed. In fact, the non-player car object's moving speed is neither base on the base speed non the global speed, but it is calculated by the two to obtain its appearance speed. When updating the non-player car object's position, it is calculated by using the equation:

$$new\_positionY = current\_positionY + global\_speed - base\_speed$$

where *global_speed* is controlled by the player, and *base_speed* is constant and based on the non-player car object's current lane. In other words, if the magnitude of the global speed is large enough, the non-player car object should move around the display screen from the beginning to the end of the lane in one tick. Furthermore, if *base_speed* $>$ *global_speed*, the non-player car object will move from position *y = 0* to position *y = screen_height* (from top to bottom of the screen), and the opposite goes if *base_speed* $<$ *global_speed*. On the other hand, non-player car objects on a specific lane can also stop moving if the global speed matches with the non-player car object on that lane's base speed. This means players can technically stay on the same lane while adjusting its global speed to the same as the current lane's base speed in order to stay alive indefinitely (this will be discussed further in subsection E. Score Tracking). Notice that the equation only changes non-player car object's y-coordinate, since that is the only coordinate non-player car object is allowed to move on (only up or down, but no lane switching). Once the non-player car object reaches the end of the lane relative to their direction of movement, it will move off-screen and the memory occupied by the car object will be freed from the program.

C. Non-Player Car Object Spawning

Each car lanes have their own similar, but independent spawning rule, with each lane can only have the maximum of 3 non-player car objects on the screen on the same lane at the same time tick.

Like its movement update, the non-player car object's spawning update is also being checked every 20 ticks. For each spawn update check, the game will first compare the base speed of each lane to the current global speed. If the lane's base speed is greater than or equal to the current global speed, the non-player car object for this particular lane will spawn from the top of the screen, while the car object will spawn at the bottom of the lane's base speed is less than the current global speed.

The game checks if there are any car objects (player or non-player) occupying the spawn area on each lane. This is achieved by flagging whether the spawning area of a specific lane is being occupied as the position of the car objects are being updated. The spawn area is one car length from the top of the screen if the non-player car objects are spawning from the top of their respective lanes, and bottom if they are indicated to spawn at the bottom.

Finally, the game will randomly pick a random number. If the number picked is the multiple of 30, the game will then check if each of the specific lanes is housing the maximum of 3 non-player car objects. If all the requirements are passed, the game will spawn in a non-player car object on that specific lanes. Again, the spawning rule for each lane is independent, meaning that two distinct car objects can spawn on different lanes at the same time frame, with one spawning at the top of the screen while the other one spawns at the bottom.

D. Car Objects Interaction

The collision detection in this game is based on a non-player car object clash onto player car objects. Since the game updates the player car object's movement position first and then update the non-player car object's position, one method of detecting collision (in which implemented onto the game) is to save player car object's new position as a global variable, then check if each non-player car object's old and new position changes should trigger the collision. Several conditions are required to be satisfied before collision detection is being triggered, given each non-player car object is on the process of position updating:

1) Collision can only occur if the player car object's lane position indicated by global variable should be the same as currently processed non-player car object's lane position
2) Rise collision trigger if the current non-player car object's graphic representation is overlapping with the player car object's graphic representation.
3) Rise collision trigger if the current non-player car object's new position will surpass the player car object's position.

Once a collision is detected, it should raise the *game_over* flag since the game is over as soon as the player car object collides with any existing non-player car objects.

E. Score Tracking

To incentivize players to challenge themselves, a scoring system is presented to reflect the decisions players have made during their gaming experience. The game will passively decrement the score by 5 points for every one player update tick (5 ticks). Players can gain back 30 points by switching lanes and can only gain back points that way. This is to encourage players to constantly switch lanes, since as discussed previously that player car never collides with any other car objects as long as it stays on the same lane and adjusts the global speed to be the same as the current lane's base speed. In other words, players are free to do that, but the scoring penalty will be huge.

IV. Game Mechanic and Graphic Representation

The game presents the game environment in a top-down, bird-eye perspective. The 3 car lanes on the left side of the screen are game spaces car objects spawn and move on. The right side of the screen keeps track of the player's current global speed and score. The player car object is represented as a small red square, while the non-player car objects are represented as small green squares. Upon game over, all movement and player control on the screen will stop responding, and a big, yellow *Game Over* will appear at the top middle of the screen.

As discussed before, one of the other main challenge when implementing this car racing game is to balance between game mechanics and graphic updates. While creating a game in which multiple game objects are moving simultaneously can result in many interesting game interactions and mechanics, the OLED Breakout Board's pixel updating speed limit how many game objects can be visually "moving" at the same time frames. Since the Breakout Board and Launchpad are two completely different components, and the method of communication the game uses is I^2C, the application would not automatically wait for the Breakout Board to finish updating indicated pixels before go on and execute the next line of code. In fact, if there are too many pixels updating on the screen at the same time, the overall updating speed will dramatically decrease, which further lag the synchronization between the graphic representation and the actual running code, and greatly diminish the quality of the game.

A. Limitation on Car Objects

One solution to reduce visual "movement" on screen is to lengthen the tick interval for a non-player car object to update to 20 ticks. This is to relax the board's workload over the average of the overall playtime.

Another solution is to limit how many car objects can exist on the screen at a time. More specifically, the game limit at the maximum of 3 non-player car objects existing on the same lane at the same time, or in other words, a total of 10 car objects with 3 separate lanes including the player car object itself.

Reducing the size of the car object's visual representation can also help with decreasing the number of pixels the board needs to update.

These strategies are effective because the mechanic of this game requires each car object to execute constant visual

"movement". However, the player car object is a special case (with an update interval of 5 ticks instead of 20 ticks) because the player would feel less engaged and less in control of their own car object if that car object has a slow response rate. That is the part where the game mechanic needs to be balanced with the graphic representation.

B. Limitation on Numeric Graphic Update

As previously mentioned numerous times, the game also tracks and displays the current global speed and the player's current score. With the global speed being updated at most every 5 ticks (global speed change according to player's input) and player's score update every tick, the board will need to deal with constant update workloads even when there are no non-player car objects spawned on any other the 3 lanes. Therefore, another method to reduce the board's workload is by updating the global speed and the score less frequently while still making sure that the speed and score displayed reflect actual information the game is operating on.

To do that, the game only updates its global speed visually when the speed is at some multiple of 10, and update the score visually when the score is at some multiple of 100. This significantly reduces the rate of communication between Launchpad and Breakout Board, especially when most of the time players would only slightly change the global speed to avoid a dramatic change of speed from the non-player car objects, and that the score's update requirement is even harsher than the global speed's. The downside to this reduction method is the loss of information to the player, although these update requirements seem reasonably clear for the players to interpret the game state they are in, while the number displayed still reflects the actual value stored in the application.

CONCLUSION

This paper presented the implementation of the car racing game, what components are used to implement this game, and more importantly, what are some of the challenges when integrating these hardware components into the implementation.

Some of the game mechanics this paper have discussed about the game are the player control, non-player object's movement and spawning rules, how the player's game object interaction with non-player objects and the game environment, and how the game's respond to the resulted interactions. Much of the game mechanics implemented within this game are aimed to enhance the player's gaming experience. For example, the distinct base speed for each lane is to introduce variation into the game; the change of each non-player car object's appearance speed based on the current global speed and their respective base speed is inspired by driving perspective in real life and also an attempt to introduce challenges into the game. However, the challenge presented to the programmers is to come up with a workaround to hardware limitation while staying true to the original game ideas, or sometimes needs to alter the idea a bit to suit the hardware's functionality. For example, implements the numeric graphic updates to base 10 or 100 instead of updating by base 1 like the actual stored value.

In conclusion, making an application with hardware limitation is a challenge that can be observed not only from the industry level but also within personal projects. The making of this game has hopefully been revealed a small part of this challenge.

REFERENCES

[1] "CC3200 SimpleLink TM Wi-Fi® and Internet-of-Things Solution, a Single-Chip Wireless MCU," Texas Instruments Incorporated. San Francisco, July, 2013.

[2] "C Programming," Tutorialspoint. [Online]. Available: https://www.tutorialspoint.com/cprogramming/index.htm. [Accessed: 09-Jun-2020].

[3] "Stack Overflow," Stack Overflow, 2008. [Online]. Available: https://stackoverflow.com/. [Accessed: 09-Jun-2020].

[4] S. Jain, S. Goel, D. Singh, and S. Baranwal, Eds., "C Programming Language," GeeksforGeeks. [Online]. Available: https://www.geeksforgeeks.org/c-programming-language/?ref=leftbar. [Accessed: 09-Jun-2020].