Monster Truck DDR Simulator

Allison Channic and Nathan “Nate” Tisdale-Dollah
University of Illinois at Chicago
CS 362 - Computer Design
Fall 2015

Table of Contents

Introduction

Monster Truck DDR Simulator is a spin off of the popular arcade game Dance Dance Revolution (DDR). In traditional DDR, two users step onto a platform with four arrows - up, down, left, right - on it, each of them having touch sensors. Both users have screens in front of them that tell them which arrow to press with either of their feet. The arrow(s) that each user is supposed to press is highlighted or sticks out on the screen in some fashion. Each time the user presses the correct arrow, that user's score is incremented. Both users keep pressing arrows until a timer runs out. Typically, this is done to the beat of a song to simulate dancing. The timer is usually dictated by either of the players before the game begins or is dependent on the length of the song. Whoever has the highest score once the timer runs out is the winner of the game.

We decided to simulate this via a computer game. Rather than having a platform or separate mechanism with a screen, we decided to create a program that two players would run locally on their machine. Like DDR, arrows would be displayed on the screen, but users would press each arrow with their fingers as opposed to their feet. This assumes that both players are running the program on a machine with a keyboard.

Before the game begins, both players would "connect" to each other through Wi-Fi, assuming that they are on the same local network. This is what allows the two users to compete against each other. Both users then agree on a time limit for the game. Once the timer begins, an arrow is displayed on the screen. If the user presses the correct arrow, their score is incremented. If the user does not press the correct arrow, their score remains the same. In either case, a new arrow is randomly chosen and displayed on the screen. This repeats until the timer reaches zero. Whoever has the highest score at the end of the game is the winner.

To add to this, we decided that while the users are playing, each of them would have an Arduino plugged into their computer via a USB cable. Connected to this Arduino circuit would be a mechanism that has a toy monster truck attached. The winner would have their monster truck released such that it pushes into the other monster truck. This is to simulate a typical monster truck derby in which a number of drivers driving giant monster trucks ram into each other and often climb on top of other drivers via their monster trucks. Again, this assumes that both players are sitting across from each other with their toy monster trucks positioned accordingly.

Purpose

Monster Truck DDR Simulator was meant solely for entertainment purposes. DDR, the arcade game this application is based off, is also for entertainment purposes, as are monster truck derbies, despite how dangerous the latter may be. There are a multitude of games, typically video-game related, in which two users perform an action such as a series of button presses and the user with the correct number or order of button presses has another external event happen.

For example, the popular video game series Mario Party has mini-games (short multi-player games within a larger video game) where all users competing must press a button on their controllers as many times as they can or press a series of buttons in the correct order. Whoever presses that button the most times, or presses the listed buttons in the correct order the most times, is the winner. Typically the winner is rewarded with coins that can be used within the larger context of the video game, outside of that particular mini-game.

Specifically, in the game Mario Party 2 for the Nintendo 64 Console, the mini-game "Mecha Marathon" requires users to push the "A" and "B" buttons on their controllers repeatedly in a game-defined time limit in order to power up fictional characters in the mini-game that will be racing. Whichever user presses the "A" and "B" buttons the most times before the timer runs out has their character win the race. After the timer runs out, all of the users simply watch the characters race, without having to press anymore buttons. Because there is no feasible way for users to keep track of how many times the other users are pressing these buttons, and because button presses during the race have no effect, the winner of the race is unknown to all users until the very end of the game when all of the fictional characters have finished their race.

Monster Truck DDR Simulator has a similar purpose. Both users must press a series of buttons correctly as many times as they can. Because both users would be sitting adjacent from

Figure 1 - Screenshot of the Mario Party 2 mini-game "Mecha Marathon." Users press two buttons simultaneously as fast as they can for a set amount of time. Whoever presses these buttons the most times has their character win a race.

Figure 2: Controller for the Nintendo 64, the console used for Mario Party 2 and the mini-game "Mecha Marathon." Users must press the 'A' button (blue) and the 'B' button (green) back and forth as many times as they can. This powers up their fictional character which is then used to race other users and their characters. Source: http://www.8-bitcentral.com/images/nintendo/n64/controller.jpg

each other, assuming neither user cheats by sharing their score with the other user, there is no feasible way for the user to know if they are currently winning. Thus, the outcome of which monster truck will get pushed forward and "destroy" the other monster truck is unknown until the very end of the game when the timer runs out. One advantage of our program compared to these other mini-games is that the user can see the results of their win or defeat in real-time, as the winning monster truck would be released almost immediately after the timer ends. The user does not have to wait to see how their reward, or lack thereof, for winning or losing respectively, affects them in a separate context nor do they have to wait a prolonged period of time. Thus, Monster Truck DDR Simulator serves as a quick and simple way for two people to entertain themselves.

Design

Designing Monster Truck DDR Simulator required designing and implementing two parts: the DDR game program to be run on both users' machines and the arduino circuit that would release the winner's monster truck. In order to establish the connection between two users, we also needed to figure out how to communicate the number of arrow presses, or score, of each user for comparison to determine which monster truck would get released. We assumed that this could be done fairly easily, given our initial assumption that in order for the game to be played correctly, both users would have to sit across from each other, thus using the same Wi-Fi or Internet network. We did not take into consideration the case where two users in such close adjacency would be on separate networks because this is such an anomaly and wasn't as important in the greater context of the successful completion of this program.

In any case, given our extensive knowledge of programming and lack thereof of Arduino, circuitry, and computer hardware, we decided to tackle the DDR program first. We initially wanted to write the program in Java so that cross-platform capability was not a concern for us, and because we felt that this was the language we were the most proficient in. Java also has an extensive collection of libraries for creating graphical user interfaces (GUI's). Our idea was to randomly generate a sequence of numbers between 1-4, each number corresponding to one of the arrows on a traditional keyboard (up, down, left, right). These arrows would then be displayed on the interface. The user would then have to enter the arrows in the correct order into a text box also on the interface. Upon each successful completion of entering the correct sequence of arrows, the user's score would be incremented by 1. At the end of the time limit, both user's

scores would be compared. Whichever user had the highest score would have a signal sent to their Arduino to release their monster truck.

For the Arduino circuit, we wanted to have each monster truck attached to some sort of latch mechanism. When the Arduino received the signal to release the monster truck, the latch would push forward or release the monster truck with enough force to be pushed into the monster truck across from it. Ideally, we wanted to have the Arduino and all required circuitry contained within a box-like structure with the latch protruding outward. The Arduino would already be connected to the USB cable and the box would have the USB cable sticking out. This way, all the users would have to do to play would be to plug the USB cable into their computers and attach the monster truck to the latch. When the game completes and a monster truck has been released, if the two users wanted to play again, all they would have to do is re-attach the monster truck to the latch. The box-like structure would allow for the two users to play the game with ease without handling any complicated computer hardware. It would also protect the Arduino and all required circuitry for the game as well as make it more aesthetically appealing to those playing or watching the application.

Implementation

DDR Computer program

We first set out to build the DDR computer program. Initially, we began to write the program in Java using the JApplet library due to its simplicity. We then realized that using C# and its WinForms capabilities with the Visual Studio IDE, we had much more freedom in creating a user interface. We were able to customize our user interface with images and graphics

with ease, given that C# comes with libraries to manipulate WinForms, and designing the WinForm itself simply requires dragging-and-dropping the desired elements in place with Visual Studio.

Once we had our user interface designed, we worked on the back-end functionality. To make things easier on ourselves and to save time, we decided to display one arrow at a time as opposed to a series of arrows. This required less GUI elements and less work. All four arrows have four states; currently displayed, correct, incorrect, and not displayed. In the currently displayed state, that is the arrow the user is supposed to press. If the user presses that arrow correctly, the arrow changes to "correct," by changing color to green to indicate to the user that he/she pressed the correct arrow. If the user does not press the arrow correctly, the arrow changes to "incorrect" by changing color to red to indicate to the user that he/she did not press the correct arrow. All other arrows are simply not displayed. After the user presses an arrow - regardless of whether that arrow is correct - a new arrow is randomly chosen and displayed. This may or may not be the same arrow that was previously displayed. When the user presses the correct arrow, his/her score is incremented by one. This repeats until the timer has run out.

Overall, the program design and implementation went smoothly. The only major issue we ran into was this odd error when overriding the "OnKeyDown" event handler. When the game began and the first arrow displayed on the screen, after the user pressed that arrow, nothing would happen. We tried refreshing the panel by minimizing the screen and then opening it again, which fixed the issue. We researched how to fix the issue by Googling "C# OnKeyDown event handler not working" and found that our problem was that we did not set the key preview

property of the WinForm to true. This is what causes the program to receive all key up, key down, and key press events. We made this adjustment and everything worked fine.

Figure 3: Home screen of the computer program. User presses "Start" to start the game or "Instructions" to learn how to play the game

Figure 4: Instructions for playing Monster Truck DDR

Figure 5: User enters how long they'd like to play for (in seconds)

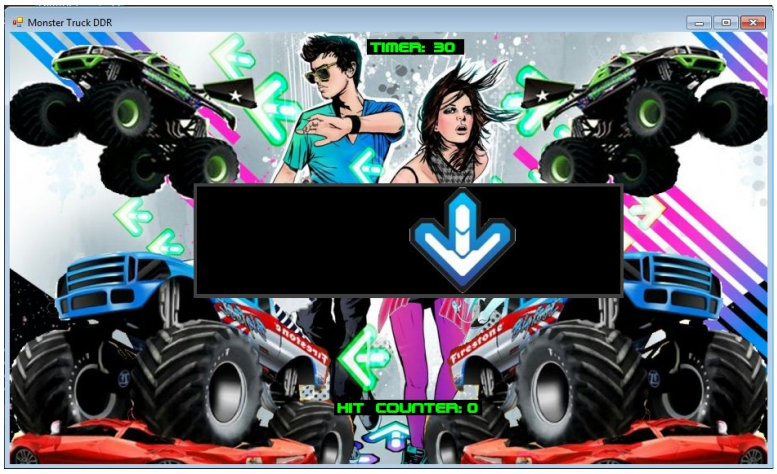

Figure 6: Screenshot of the actual Monster Truck DDR game. The arrow displayed is the arrow the user must press in order to increment his/her score

Monster Truck Propulsion Circuit

Once the DDR program was finished, it was time for us to build the circuit. Going off our initial latch mechanism idea, we researched circuits for this to see what has already been done. This actually seemed to be more complicated than we originally thought. There was no simple latch piece we could buy (or if there was, we didn't know how to find it) that was small enough to simply propel the monster truck forward, and all of the resources we found were catapult-like

structures that were too big for our needs and required a great deal of wood, parts, and more time than we had to finish our application.

We then decided alternative methods to propel the monster truck forward. The best idea we found was to use a solenoid: an electronically controlled spring. When the spring propelled forward, we believed we could redesign our original idea so that it would push the monster truck forward with it.

After researching how to construct an Arduino circuit that would control a solenoid, we came across this great tutorial that came with instructions and a schematic on how to build the circuit that would do so. The instructions came with a list of parts necessary to build the circuit. With this in hand, we bought the parts listed as well as two toy monster trucks. When buying parts, the two biggest concerns of ours were price and functionality. We did not want to spend too much on parts given the short-term duration of this project and we did not want to buy parts that were not specified in the circuit tutorial just in case we could not get them to work. We got the cheapest parts necessary, including the monster trucks and hardware components.

Figure 7: The two monster trucks we bought for this project. These were the two cheapest monster trucks at Target that weren't too small nor too big.

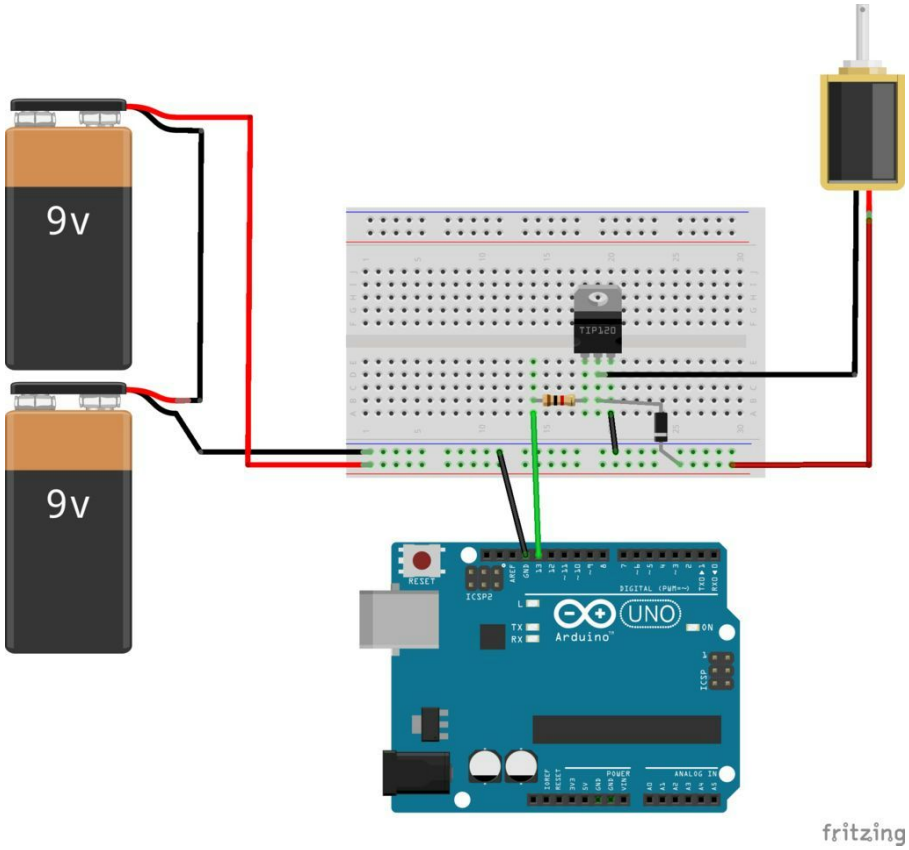

Figure 8: The schematic for our solenoid circuit that would be used to propel the monster trucks forward. Source: http://www.instructables.com/id/Controlling-solenoids-with-arduino/?ALLSTEPS

Materials and Components

For one user, the following materials were required:

- Arduino Microcontroller (1)
- Breadboard (1)
- 1K Resistor (1)
- USB cable (1)
- Jumper cables (3)
- 12V Solenoid with leads (1)
- Toy monster truck (1)
- TIP 120 Transistor (1)
- 9V Battery (1)
- 9V Battery Connectors (1)

- 1N4004 diode (1)
- Cardboard scraps
- Computer
- Keyboard

Combining it All Together

Once we had all of our parts, the first thing we did was build the circuit. We went straight off of the online tutorial. Essentially, the circuit is two-in-one: the solenoid is powered by the battery, but requires the use of the power from the Arduino to obtain the power transmitted by the battery. The wires, transistor, diode, and resistor transfer this power accordingly to the solenoid via the breadboard. The battery caps attached to the battery allow for the power from the battery to be transmitted. To test our circuit, we wrote Arduino that propelled the solenoid back and forth every second.

Our next step was attaching the solenoid to the monster truck. We could not leave the solenoid as is because it was way too short to touch any part of the monster truck except for the wheel. If the solenoid were to propel in this way, the path of the monster truck would be skewed. We didn't have a particular design in mind, so we just cut up some cardboard in the shape of a box to fit the solenoid. We made a slit in the back of the box so that the leads from the solenoid could be plugged into the breadboard. To attach the box containing the solenoid to the monster truck, we cut out the small, plastic backing on the monster truck, added more cardboard padding to the bottom of the box, and stuffed the bottom of the box into the back of the monster truck. From there, with the solenoid in the box, we plugged the solenoid into the breadboard and tested.

Even though the solenoid was pressed up against the monster truck, it was unable to push the monster truck. We tried re-adjusting the solenoid so that just the tip was touching the back of the monster truck and again so that the solenoid was pressed in some. Both just caused the solenoid to tap against the monster truck without moving it. We then tried increasing the voltage going to the solenoid from 9V to 18V by adding another battery and twisting the wires on the two battery caps. This did not make a difference. It was clear to us that the 12V from the solenoid was not enough to push the weight of the monster truck. If we had used a solenoid with higher voltage or a monster truck that weighed less, odds are the mechanism we built would successfully move the monster truck. We did not, however, have the time nor money to invest in new parts, so we decided to leave things as-is and continue getting as far as we could in making the application.

Our next step was to try to communicate between two users. Our initial assumption was that if the two players were sitting directly across from each other, they would be on the same network. We also assumed that if two users were on the same network, we would not need a central web server to communicate between the two users. Turns out we were wrong, one does need a central web server for any sort of communication across CPU's. We did not have the time nor resources to make a web server (where would we host it? Which player, if any, could or should be the "host"?), but we did research different C# API's and the .NET framework to see how this could be done.

To allow for multi-player functionality, we modified our existing program so that two users could play on the same machine. Player one would use the standard arrow keys (up, down, left, right) and press whichever arrow was displayed on the top-most arrow panel while player

two would use the ‘W’ (up), ‘A’ (left), ‘S’ (down), and ‘D’ (right) keys for whichever arrow was displayed on the bottom arrow panel.

Figure 9: Screenshot of modified DDR Program to allow for two-players playing on the same screen/machine

This assumed that whichever machine the two users were playing on had two available USB ports while the game was in progress: one for each Arduino that powered each monster truck. We then ran into the issue of how to determine which Arduino would be released based on which player won. We tried setting up two port objects in the C# code, which is how we were able to propel the solenoid for a single circuit, but we were unsuccessful. We tried debugging this by making it so that only either player’s solenoid would be propelled regardless of who won, but one of the Arduino’s consistently did nothing even when it was told to execute the propulsion of the solenoid.

Alas, our final project is Monster Truck DDR Simulator meant for two players using the same keyboard and machine, but only works for one player, assuming that the number of arrow presses for the user is greater than zero.

Figure 10: Cardboard box to contain the solenoid and attach to the monster truck.

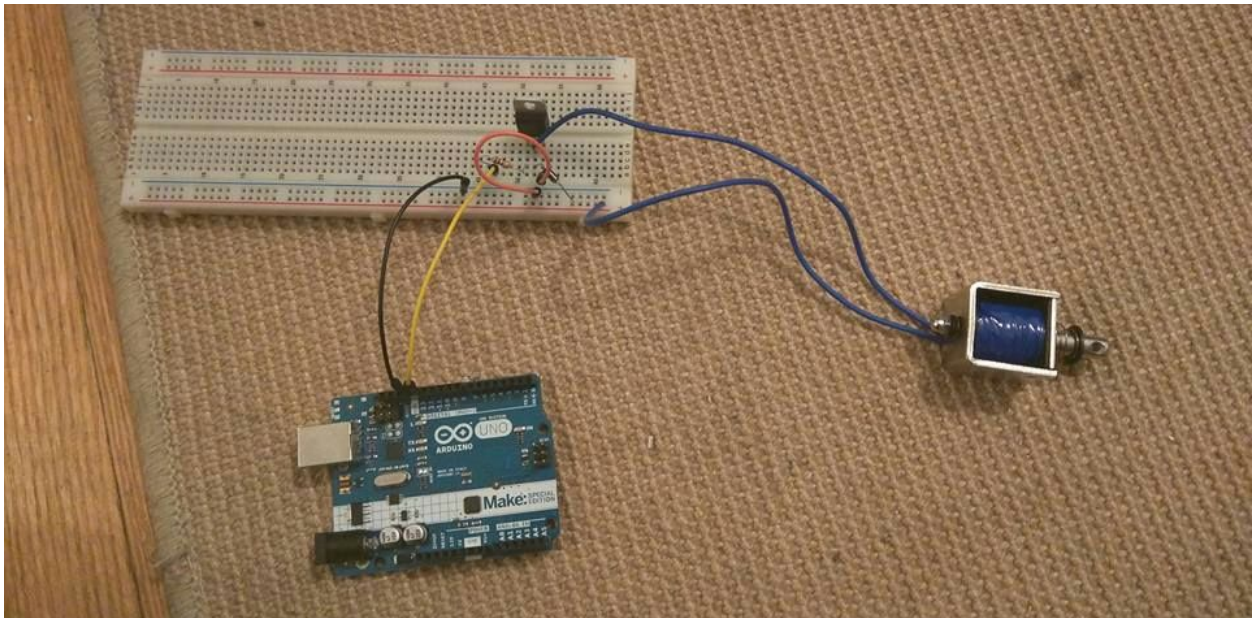

Figure 11: Close-up view of the circuit used to propel the solenoid

Figure 12: The final circuit with the solenoid attached to the breadboard and the monster truck, contained in the cardboard box. The Arduino is plugged into the user's computer via a USB cable.

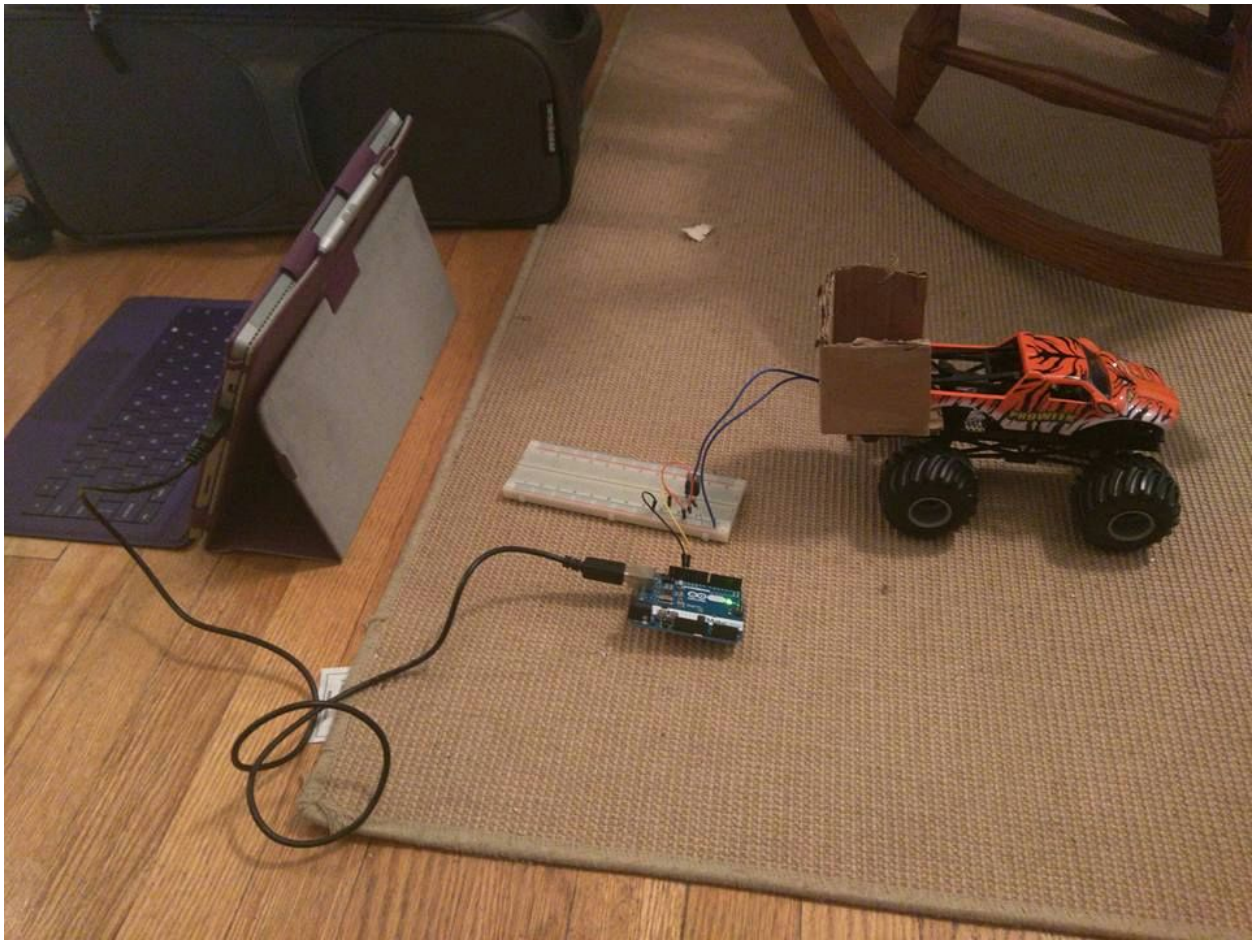

Figure 13: The circuit plugged into a computer with the USB cable.

Execution Instructions

1.) Make sure the circuit and all of its components are set-up and plugged in properly, including the solenoid which should be in the cardboard box. Cardboard box should be attached to the monster truck.

2.) Plug in the Arduino via USB cable

3.) Run the MonsterTruckDDR executable file

4.) Click the start button

5.) Enter a time limit you would like to play the game for, in seconds

6.) As player one, on the top-most arrow panel, press the arrow that is currently displayed by hitting the corresponding arrow key on the keyboard

7.) Repeat until the timer hits 0

8.) Solenoid should release and hit the monster truck

Future Expansion and Improvements

The biggest improvement that could be made to Monster Truck DDR Simulator as it stands is to modify the circuitry so that it has enough power to push the monster truck. This would require obtaining a solenoid with a much higher voltage as well as a battery with enough voltage for this solenoid. One could also use multiple batteries and connect them twisting together the wires on the battery caps. Another alternative would be to get monster trucks that weigh considerably less so that the circuit, solenoid, and amount of voltage as it stands is enough to propel the monster truck while the solenoid rests in the cardboard box.

Ideally, the solenoids would have longer leads so that the monster truck could be pushed forward to a further distance without either of the users having to worry about disconnecting the solenoid and then having to put it back into the breadboard.

The circuit could also be contained in a box-like mechanism to protect the Arduino and its corresponding wires from the elements. The structure would have to allow room for both of the solenoid's leads to protrude. In order for the solenoid to push the monster truck, one idea would be to have the solenoid attached to the box, one end of the solenoid being contained within the box to allow for adequate movement, and simply align the monster truck with the solenoid. Another idea would be to modify the existing cardboard box that the solenoid currently sits in to make it sturdier and permanently attached to the truck.

In order for the two-player functionality to work as intended, one must set up a web-server. When the timer runs out, both users' scores would be sent to the server and a simple comparison would be done to see which user has the higher score. The server would then execute the corresponding code to push that user's monster truck forward.

References

Solenoid circuit (including necessary parts):
http://www.instructables.com/id/Controlling-solenoids-with-arduino/

Mario Party “Mecha Marathon” mini-game reference:
http://www.mariowiki.com/Mecha-Marathon

Code we referenced to send C# value to Arduino and use that value in the Arduino code:
http://forum.arduino.cc/index.php?topic=39811.0

Stack overflow page that solved our weird refresh error with the WinForm:
http://stackoverflow.com/questions/8644083/keydown-in-c-sharp-doesnt-work-for-some-reason