Easy homemade pinball machine

More than 700 cheap, natural substitute recipes for everyday branded products. Divided into 3 sections: In the Kitchen, Creating Health & Beauty Products, Making Products for the House & Garden. Fill your cabinets with home-made, super-high-quality products that work great — without all the chemicals and preservatives you've added. Find simple, money-saving recipes for food staples, cosmetics and health items, household compounds, cleaning products, pet products, garden products and more! Special section on how to maintain a well-equipped kitchen. Format: Hardcover Dimensions: 7 x 9 SKU #: 51740 ISBN-13: 978-1-62145-416-8 Publisher: Reader's Digest Release date: 2008 Pages: 400 If you're like me, you love pinball but don't have the money to buy or place to match the full-size game. So why not build your own? Here, we'll walk around as you create your own custom pinball game powered by Arduino. The game has lights, sounds, features real pinball parts including bumpers, drop targets and slings, and even has a ramp. This project requires a very large amount and variety of material, so consult each subsequent section to get the new materials you need to complete each step. For starters, this is very helpful if you have access to a laser cutter or CNC router, as well as basic electronic and hardware toolkits. Author note: This instructable was recently published and not all project and software files have been completely organized. If you plan to use our files, leave a comment so that we can make sure that everything is in the most current condition. Pictured above is the Solidworks playfield design and auxiliary assembly. The playing field is purely custom, but the shot lines (such as the back looping shot curve) are designed based on real pinball machines to ensure smooth play. One of the difficulties is that due to their complexity, the actual pinball parts (e.g. bumpers and drop targets) were not modeled, but it is still necessary to ensure that everything fits under the playing field - the parts are much larger underneath than above. The files are contained in the repository, so you can customize the project to suit your admonitions. A few design highlights: The playing field is 42 by 20.25 inches, exactly the size of the 1980's Bally style of play. It is made of 1/2 plywood, which is standard and should not be changed because pinball parts assemblies are designed for this thickness. The walls here consist of a layer of 1/2 on top of layer 1/4. Only 1/2 walls were included in the first prototype, but they proved too short and could jump out of the pinball into the air on particularly strong shots. Secondly, this design allows for a slightly raised line of the shooter (pictured above), which allows the ball to drop slightly into the playing field but does not fall from the The ramp is designed with transparent acrylic and 3d printed brackets. Crosses the playing field so that it gives the player the opportunity to hit the ramp several times in a row from the left side As such, clear acrylic is used not to cover the player's table view:Finally, the playing field is supported by short walls in four corners that keep the playing field in a standard 6.5 degree slope. The back wall has a bottom shelf that can be taken out and is used for electronics assembly. This results in a game with a full-size playing field, but it is much more compact than a typical game and can be played manually by one person. Since the playing field is standard size, however, these supports can be removed if you want to put the playfield in a standard pinball cabinet. To do this, consider adding a ball return team that is not included in this project. To cut out the layers of the playing field, we used a laser cutter. However, a laser cutout strong enough to cut 1/2 plywood is difficult to find, requires high quality plywood and can risk starting a fire if you are not careful. Typical playing fields are cut using a CNC router - while some corners may not be as sharp, you should still achieve decent results. For simplicity, the steps below assume that you have access to the same laser cutter that we did. There are people who have had decent results using only drills and puzzles, but you have to be very careful and very patient if you go down this path. The first step in creating a playing field is to transform the project into . DXF files that can be fed into a laser cutter. For example, a playing field . The DXF file is pictured below. The files used in this project are in our repository. With the help of a laser cutter, we cut out the shapes on the pitch, the intermediate layer 1/4 (we used duron, a cheaper material for prototyping resembling wood, but also works 1/4 plywood), the top layer of 1/2 and 1/2 support. Materials needed: 1/2 plywood for the pitch and base 1/4 plywood or duron for the intermediate layer of the wall 1/2, 3/4 and 1 wood screws Access to the CNC router or laser cutter Starting by clamping the elements from the duron layer 1/4 on the plywood in the right places. Using a hand drill, drill the pilot holes first with a 3/32 bit, and then use a flat head 3/4 wooden screws to attach a layer of 1/4 to the playing field It is important to do this from top to bottom (ie, so that the screw first passes through the layer 1/4, and then to the base 1/2, because 1/4 of the parts are small and thin and bend from the base layer if drilled in the opposite direction. It is also important to make sure that the screw heads are flush with the 1/4 layer and do not provide additional thickness. Last note: these bolts can go almost anywhere, as this layer will mostly be invisible to the player when the playing field is folded. But there is an exception - do not put screws into the shooter's belt. (We initially made this mistake.) Then attach the sidewalls and use the longest wood screws to drill into them from the top of the slab, again screw heads are flush with the top. When you do this, press down on layer 1/2 on top of the duron and as before, except for this time of screwing in from below with screws 1. Since the top layer is 1/2 thick, it is less likely to bend from the base, and screwing in from below ensures that the screws remain invisible to the player. Finally, attach the shooting block (pictured above, with the shooter) by screwing on from the bottom side with 2 screws so that the block cannot turn easily. The shooter block has a U-shaped slot that fits the shooter, which can be installed by tightening the nut on the other side. It may also be the use of lubricant to reduce friction between the firing rod and the ball. The project may require some adjustments at this point. For example, in our project, the cutting for dropping purposes was too narrow and had to be extended with a dremel. If you use our files as more than references, try contacting the authors who may be able to provide the updated files. It is also a good idea to sandblast down any rough areas where two pieces of wood meet. For the most part, this ends with woodworking and we can move on to introducing components. Materials needed: 3/4 flat screw wood head shooter mounting Longer (~1.5) wood drill manual drill with 3/32 bit lubricating oil 1 flat screw with wood head file and/or dremel, and sandpaperThrough this point in the design stage, you should have a general idea of the orientation required to make sure that all the elements actually fit under the playing field. (If you are using our project, please refer to the photo below our table above.) First, install drop targets, stand-up targets, and sling bands by placing 1/2 of the wooden screws through the mounting holes in the assembly. Do the same with pop bumpers, but make sure to remove the cap first, or the band doesn't fit into its hole! Secondly, you need to install flipper assemblies. Make sure they rotate in the right direction. The electromagnet, when fired, will pill the pin into the coil, and it should rotate the shaft so that the flipper rotates toward the playing field. After installing the fin assemblies, attach the fin bats on the other side. Use the wrench on the lock nut in the assembly to tighten them to the spot, and then use the spring that should come from the assembly to make sure that the fins are appalled back down when not released. Similarly, install all rollover switches with 1/2 screws, ensuring that they can be easily pressed from above and spring back into place. Using 6-32 screws, you should also attach the gate switch in the upper left corner of our design. This gate switch also serves as a one-way hole that allows shots to the right and from the shooter to fall into bumpers. This is a design aspect that causes shots to enter the right ramp and right loop to different locations and adds more variety to the To install the lights, first place the plastic inserts in their holes. These tiles are about 1/4 thick. If you are using a CNC router, the correct way to mount them is to cut out layer 1/4 1/4 larger than the insert hole. In our design, because the laser cutter can not cut partial layers, 3D printed brackets that support tiles. Use epoxy to hold the tiles in place (first roughing the edges) and sandpaper to make sure the tiles are aligned with the playing field. Then insert the LEDs into the brackets by inserting them and turning them into place. Then screw the brackets in such a way that these LEDs are placed directly under each insert. The light brackets connected below are quite thin, and in fact thin enough that 1/2 screws can pierce the top of the table. Use several washers so that this does not happen. The playing field bars are installed with 6-32 screws. Once installed, wrap the rubber rubber rubber set around them to make passive bumpers. They give the table a lot more life than if the design were to be completely plywood. Use the same screws to attach lane guides just above the fins. Also glue the end-of-game switch in place. Keep in mind that most games have a dedicated team returning the ball like the one here. However, this has not been taken into account in this project, primarily due to costs. The trade-off, of course, is that the player is now responsible for putting the ball back into the scorer's avenue when it will run down. However, we have a shooter who is attached to the shooter block, as in the photo earlier. The flipper buttons and start button are installed by placing them in holes and locking in place with palnuts. The leaf switches of the flipper button are screwed inside the buttons with 6-32 screws and will close the circuit of the switch when the buttons are pressed. At this point, your playing field will (in advance) resemble an almost complete pinball table! All that's missing is the ramp. Feel free to brag among your friends about how amazing it looks while privately being scared of how much wiring and soldering is to be done. Materials needed (most were purchased from PinballLife.com and can be found simply by searching through the following terms).1 3-bank target set 3x pop bumper assembly 1 left flipper assembly 1 flipper band 2 fins bats 2 fin buttons 2 flipper buttons 1 boot button 1 rubber ring set ~ 30 star posts play field, (1 1/16 used) 2 lane guides 2 leaf flipper button switches 2 slings bands 1 standup target 10 rollover switches 8 LED #44 bayonet style light 8 bayonet style light brackets (Miniature Bayonet Base 2-Lead Socket with long mount Ws bracket) 5 1-1/2 x 13/16 blue insert arrow 3 1 x 3/4 clear ball insert 6-32 screws (2.5, as well as some smaller sizes), nuts and washers ~ 2 wide gate switch (like this one here , it can be hard to find, we have scrapmed ours with old broken pinball ramps purchased on eBay)To ramp, use 1/4 acrylic for basic items and 1/8 acrylic for Side. Transparent acrylic will give a nice, clean look, without blocking the view of the pitch for the player. The use of colored acrylic can also be a sweet lye, a it is not recommended to use a completely opaque material, such as wood. Ramp supports are printed in 3D using a makerbot and screwed to the play field and plastic using the same 6-32 screws. Acrylic pieces here are glued together with acrylic cement, which is a solvent that essentially melts and welds plastic. Make sure you use a small amount, and this will make you a very strong bond that is almost invisible. At the entrance to the ramp, we attached a ramp flap, as in the photo above. It is a thin piece of metal that gives you a very smooth transition from the playing field to the plastic ramp, instead of the pinball you have to jump to 1/4 of the thickness of the plastic. You can buy one

of these cheaply from a pinball specialist store or Ebay (we did), or just make one of your own from the sheet metal. In commercial games, they are riveted so that the screws do not stick around and get in the way of the ball. Since we didn't have the right equipment to do this, we made sure we used flat-break screws and properly falsified the hole in plastic and metal to achieve the same effect. In the front right corner of the ramp there is a narrow gate switch, where it rotates across the playing field. This switch is what records when a successful ramp shot has been hit. Materials needed: 1/4 transparent acrylic (sheet 12x24 1/2 transparent acrylic (sheet 12x24 48V can blow up some transistors in this configuration. I recommend using 35V or lower with these electronics or using a more professional control board resource such as those listed here: This machine has 3 voltage levels: 48V for electromagnetic power, 6.3V for LEDs and 5V for logic and sound. To provide these voltage levels, we used a CNC power supply for 48V and ready-made DC adapters to provide 6.3V and 5V. (It is possible to use just 6.3V because Arduino down regulates the supply voltage to the 5V output pin, but we kept these power supplies isolated). The 48V is high voltage and although it is not fatal in itself it can be harmful to parts and can quickly cause components to overheat if there are any problems with the circuits. Use a slow 5-A fuse at both the input and output of the main 48V power supply to avoid fire if any of the transistors are short. On the Arduino disc, we attached wires with female Molex connectors that corresponded to the input and output requirements of each of the three sub-plates: an electromagnetic controller plate, an array of lights/driver sound, and an input board. In our project, we had the following pin assignments. This, of course, is quite flexible. Pin 0 remained open. (for these do not allow us to make lists of starting numbers) 0.)Open Open Interrupt / Input Active pin Encoded input pin Coded input pin Coded pin Coded input pin Coded pin Input Input Right bumper output Right bumper Output Middle bumper output Left bumper Output main Output flipper Main switch light Output Main Output Main Output Output Light Pin Output Light Pin Output Pin Pin Sound Output Audio Output OpenAlthough sound not implemented in our project, SCL and SDA pins can be used for display, and the remaining pins can be used for additional control, such as adding a function (return of a sphere) or more lighting combinations. Materials needed: 48V CNC power supply (like this) Ready-made power supplies 6.3V and 5V (like this) 5A fuses and fuse holders, and shrink tubes to connect Molex Arduino connectors prototype board disc Plenty of 22AWG wire, soldering and patienceThe driver board is responsible for rotating inputs from Arduino, flipper buttons, and the sling screen switches to firing coils. Since the signals are at 5V and the electromagnets at 48V, strong MOSFETS power is essential for signal transmission. Transistors used in this design are those 100V-rated MOSFETs from Mouser.There are three diagrams pictured above that include fins, slings, and bumpers/drop targets. Each has slightly different requirements, but in all of them, when the transistor receives a 5V signal, the current path for the electromagnet opens, and 5-8 amps are thrown through the coil to give a powerful kick. This is a lot of electricity! In fact, this current will burn out the components if the transistor is held for more than a very short pulse. Make sure that when testing this circuit using software or other methods, never fully power the electromagnet for more than a second. The main source of problems in the above circuit is an induction kick. Electromagnets are powerful induction coils, and as you know, the current in the coils can not change immediately. Therefore, when the transistor is turned off, there is still a short moment when 5-8 amps flow through the electromagnet, and everything that the current has to go somewhere. If no path to the ground is given, this current will drive the voltage on the transistor to suck up to hundreds of volts and destroy the transistor. In addition, when the transistor is destroyed, it will go crazy all three terminals, which causes the flow of continuous current amplifiers and can destroy the electromagnet if there is no suitable fuse installed. (We destroyed 8 transistors in our discovery and are trying to deal with this problem, but fortunately there are no electromagnets because we have always been quick to manually disconnect the power supply.) There are two ways to prevent inductive kicking: first, each pinball assembly should be equipped with an LED that indicates the outflow of the transistor back into power. In theory, this should prevent power supply voltage through the transistor, because once this is done, the LED will turn on and flow all the remaining energy from the inductor. Induction. in fact, these LEDs themselves do not turn on quickly enough to inhibit the induction kick themselves. To solve this problem, we added rc circuit snubber. This circuit is equipped with a serial capacitor with resistor. The capacitor absorbs sufficient current from the inductor in such a way that the LED has time to turn on and perform its function. For more information about RC snubber circuits, check here. The circuit of the bumper/droptarget electromagnetic controller is quite simple and has only a transistor, electromagnet, cuddler and connection to receive the input from Arduino. In this disc and subsequent plates, make sure that the electromagnet is wired in such a way that the LED (which is not shown in the diagram) points towards the high voltage. The flipper driver circuit is a bit more complicated for three reasons. First, to get a quick response between the press of a button and the operation of the flipper, it is recommended to create this response directly in the circuits, and not as separate inputs and outputs supported by Arduino. The delay caused by Arduino is small, but an experienced player will be able to say immediately and will be frustrated by the lack of control. Secondly, the fins are equipped with two different coils (low power coil and high power coil), a jump end switch that starts when the fins are high. This switch serves an important function of allowing a high-powered coil to fire initially to give a powerful jump, but switching to a low-power coil (~130 ohms vs. 4 ohms) that gives you enough power to keep the flipper held up as long as the button is positioned, but not draw enough current to burn out the electromagnet. In the picture below, the EOS switch is normally closed, but our team had a normally open switch and required another transistor to turn it into a normally closed signal. Thirdly, while we wanted a button to control the fins directly, we also have a master switch signal from Arduino that could activate or disable the fins depending on whether the ball was in play. This results in the use of a third transistor in the circuit. Similarly, the sling board has its own complications. Although it uses only one transistor, it, like fins, should be controlled directly by input switches (which are wired in a series) for fast response, as well as do not require additional output pins on Arduino. Unfortunately, if the transistor gate is directly connected to the switch, the reaction is too fast to have more than a barely noticeable kick, because the switch does not remain closed for a very long time. To have a stronger kick (i.e. allow the sling ing electromagnet to follow it), we added a diode and a large resistor at the transistor gate, which allows for a quick response, but creates a large constant of voltage decay time in this node, so that the gate remains close to 5V transistor on) long enough to have a noticeable kick, even after the sling sling have been reopened. Another complication is sending this entry to Arduino, because the input plate (as we'll see later) requires low inputs, and the sling gun works when the input is pushed high. To solve this problem, we have included a third transistor that closes when either input goes high and thus can be treated like any other input switch on the playfieldThe driver board (actually two boards) consists of two flipper controllers, two sling drivers and four single-switch controllers for the remaining electromagnets. Instead of soldering directly, we used molex 0.1 connectors to attach this board to electromagnets, power supplies and switches so that any repairs or adjustments can be made more easily. We used solder bread boards for our designs, but designing real PCBs with these features would have a much cleaner result and would help alleviate the mess of wires that these machines inevitably have. Materials: 12 rated power transistors 10-50 uF 10-50 uF (if possible non-circulating) 300, 5k, and 500k, and 3M resistors 1 smaller transistor for sling switch Several 1N4004 LEDs Prototype soldering boards for slicing (or, better yet, design your own PCB)Since we use only Arduino, we are limited to 20 digital pins. The pinball machine, however, has dozens of unique switch inputs, not to mention the outputs needed for light, sound and drive electromagnets. To alleviate this problem, we assumed that no two inputs would be triggered at once (thus limiting us to using only 1 ball). This assumption allows us to encode the switch inputs by converting them to a 5-bit binary register with 6 pin, which triggered a break when the correct switch input is received. To achieve this, we used a cascade of 8-to-3 encoders to make the encoder 24 to 5 using this encoder in the layout shown in the pictures above. This was one of the most important achievements of the project, because it allowed us to significantly increase the complexity of our machine with our initial plan to have fins, bumpers and one or two goals. A second prototype plate was used to place each of the 24 male Molex connectors; any switch on the playing field would have a female connector at the end of a long cable that plugs into that board. Drop targets are a unique case that can be handled in several ways. What we did was wire each drop target switch in the series so that the input is closed when they are all down and allows Arduino to send a signal to the electromagnet to fire the targets to drop back up. Materials: 4 3-state output priorities 8-to-3 encodersI use a 3-to-8 decoder to control our lights. This provided us with a limitation that we could not light more than one light at one time, but it was an acceptable compromise to release the pins to Items. We have also included a 4th master output light that can control all lights at once. This, for example, can allow us to flash all the lights of light times when the game is first enabled (giving a strong indication that something actually happens to the player when he or she presses the start button, which otherwise is difficult without a ball trough or color display). The above scheme has a transistor circuit similar to the controllers, but much simpler, because lower in-game voltages (6.3V for lights) need smaller transistors and do not require as many protective circuits. We used a diode or gate for transistors to isolate the signal of the main switch and the individual light signal. This allows us to use only one transistor per light instead of two and prevents the fight against Arduino and the kettle system to fight the source or current of the sink. While we used low-current LEDs for each of the playfield lights (those under the inserts), the start button and 3 pop bumpers were equipped with bulbs that attract about 250mA each. Transistors are rated for a continuous current of 530mA, so in order not to exceed this, we made sure that only two bulbs ever passed through one transistor. We also attached a passive 5V piezo buzzer that allows us to play basic sounds to this board. Custom light and sound sequences can be programmed using the light_sequence + sound_sequence function or through the Pinball.10 light transistors interface (we used these) 5V Pizo buzzerThere are two options for defining pinball rules. You can interact with the game using a customizable pinball document, or hard code game rules. Coded game rules allow for more flexibility, including more shots and time bonuses, while the use of the pinball/parser document system allows for more flexible but simpler rules. We'll start with an interface for a customizable game and then detail some of the coded rules of the game so you can choose the configuration you want for your own pinball game. See the github repository here for the files referenced in this project. Part 1. Design your rules of the gameAlce not always a state device for a pinball game is pictured. This is specified in the default startup code. Now you have two options - either write your own code for the machine, or use specific formatting for the pinball game. In pinball-text, you'll find three sections: one for parts, one for states, and one for activities. Here you can define specific actions for each component. For most components, you'll probably want to stick to a single-state machine. For example, if every time a bumper is hit, the player should score 100 more points, light the ramp light and score 100 points, the status diagram will look like figure 1 with the corresponding code. If you want a component to have a multi-state machine, say, you wanted to turn on the light when the bumper is hit, and then turn off when it hits again, diagram / the corresponding states will look like Figure 2. Our specific machine provides as in Figure 3, for which you can define rules. Their names, internal encoded macros (which you don't have to worry about, but can be useful if you decide to examine the source code) and break codes are given in Figure 3. Figure 4 combines these names with the game box components. Tips for writing a pinball game Because game components are associated with specific interrupts (indicated by the pos field), which in turn are defined by hardware, we do not recommend modifying the parts section too much outside the states box. We suggest that you keep the status 0 and action 0 for components that do not affect the score, such as the start button and the game switch. Our code looks like figure 5. The eight lights on the board are controlled by a set-top box 3 to 8 + one main switch, as described earlier. Specific lights can be lit by saving pins corresponding to the binary encoded version of the high part code. The Helper light_sequence provides the user with an interface to specify the light they want to light, and macros are defined in the state_machine_headers.h document. The table has been made available again for programming convenience. As for Sound, we used arduino's tone library to program short sound sequences for various in-game events. We have four pre-made sounds that you can select (using executeSound (< # sound you want>)). These sounds correspond to a long, cheerful sequence, a short joyful sequence, a short sad sequence and a long sad sequence. If you want to program your own sounds, you can search here for how to it (pitch.h is included in the repository): and you're done writing FSM, here's how to load the game on Arduino (assumes you're using a Mac). All files can be found in the github repository. Unzip the arduino-serial zip file. Go to the arduino-serial file and save the game configuration file here. Pinball.txt contains a sample template that you can use. Open Arduino. Submit a pinball game sketch. Open the terminal and type the following commands: make./arduino-serial -b 9600 -p pinball.txtNow, we should read and store the data in Arduino's internal memory. If there are any distorted lines, Arduino will print an error message and you can resend the file. Once you have finished transmitting the code using the terminal, for example, when Arduino prints a finished message, you can open Arduino Serial to read messages from the game in progress. Common issues / optimizations for a software gameDiartart vs. configurable games - we noticed that interruptions in an encoded game responded much more accurately than in a configurable game. This may be because the customizable game had many general-purpose features that required conditional instructions. This slowed down the speed of the loop reading, which caused us to miss a few interruptions and affect the overall speed of the game. In order to resolve this we have reduced some of the possibilities of customizing the config file game to achieve acceptable response times in the circuit. Initially, we had concerns about Arduino's RAM capacity and how much game rules it could store, but this turned out to be a smaller problem than originally expected, and it was the speed of the loop that was a bigger limiting factor. Debouncing interruptions - due to the quick action of the pinball game, we had several cases where the interrupt pin received several interruptions for pinball hitting only one component of the game. Additionally, because these interrupts were received before the encoder had time to correctly read all the input, the interrupts would be linked to incorrect components. To resolve this issue, we used an external debouncing library that corresponds to 1ms when the first interrupt is received, giving time for the encoder pins to reach a high level before the game reads the input code. Display - Although the serial display allows the game to print detailed messages, it is difficult for the player to read the output messages while playing a fast pinball game. It is also bulky for the player to play games with a connected computer. In the future, we hope to implement a digital display that can display the result and other game information on the display that the user can easily see, such as an LED sensor or a 7-segment display. First, read the state_machine_headers.h document to understand the global data structures that store state machine information. Before loading into Arduino code, you must initialize these data structures into the rules of the game in the Arduino IDE environment. The following data structures are provided:Game structures to store information about each part of the states to store information about the status transitions actions to store information about actions to be performed\ These structures are populated by a read file. Define inputs/outputs for all pins. Break pins should be defined as INPUT pins. In the main loop, check each cycle to see if a break has been fired for each game component. Define each game component in the switch statement. The executeState helper function updates the current state of the part and performs actions based on encoded information. The on-site encoded first version of the game code can be found in the simplepinballgame.ino fileTo connect Arduino to our driver boards, we used protoshield to make it easier to access pins on other boards. There are many wires, so watch out! Follow the layout provided in the electronic pins and layout to connect your Arduino slots to the corresponding pins. Molex connectors should help a lot in determining which connectors they connect to. Here are some quick troubleshooting questions if you encounter any of the common issues we've done: None of the inputs is registered by Arduino! The nature of the input cewar is that there are 6 input pins to Arduino: 5, which shows which input is triggered, and 6 pin, which is high if a single input is triggered. Written code only detects when this sixth pin changes from low to high. So if Arduino doesn't receive any inputs, and you're sure all or at least most switches work, check if any switches are locked closed. For example, if all drop targets are down and have not been fired back, this is a closed switch and prevents Arduino from receiving other inputs. The shooter is really poor! Check that the nut that keeps the shooter in place is fully tightened, or that the shooter block is not loose. Alternatively, oil rod shooter. Some inputs, such as rollovers, are not always recordedIt can be a mechanical/design problem if the switches are placed in too wide a belt, allowing the ball to be around them. If, for example, you are busy playing the tone using the tone library and delay() statements, Arduino will not be able to receive the input at that time. One of the solutions we used was to play only the sounds of the ground shot, the standup target, the start button, and the end-of-game switch, as we knew about how much time we would have after these shots before the new input could be launched. The wrong bumper shoots when the ball is switched! OR bad lights are coming at night! It is true that you do not assign specific headings for specific lights or specific electromagnets, which means that the first time you connect everything (or subsequent times if you do not somehow mark them), the output pins (or the encoding of the output light) are connected in any order. Use trial and error to recognize which pins correspond to which output and adjust the code accordingly. For lights and bumpers, it's not that bad - but it definitely means all the inputs and saves, which is what can have up to 24 values and will take a bit longer to calibrate. The inputs are fine when I press them, but sometimes they are wrong during the game? The encoder has the unfortunate property of sometimes pulsating the pin pointer high before the 5 embarki pins have fully resolved their values. We knew this was the case when the number of the switch you pressed was switched off after one, but it might appear differently. We solved this problem by using debouncing to create a small piece of latency between when we notice that the switch has changed and when we record which switch it was. Be careful though, because too long a delay (more than 15-20mS) can cause you to completely miss the input. There are so many wires and I can't keep them all organized please helpSorry, but i haven't really figured out a good solution for this yet. Yet.