Technical Manual
Science App for Lewisburg Children's Museum

The Three Musketeers

Michael Hammer · Kenny Rader · Keyi Zhang

(mph009@bucknell.edu · klr020@bucknell.edu · kz005@bucknell.edu)

April 26, 2017

Table of Contents

Initial Project Description

Project Description

The Lewisburg Children's Museum would like to have a launch-sequence game or app for the "Mission Control" exhibit where children can enter a series of parameters related to a rocket and attempt to "launch" it. The requirement is that the students need to be able to learn from making a prediction and attempting the launch, having the launch "fail" for explicable reasons, re-adjust some parameters and eventually make a successful launch.

Goals

Provide an interface for children to learn about physics, rocket science, etc. through play and engagement with the feedback. The display needs to communicate to preliterate children. The app or game should function to direct children to use the "sequence" and follow simple instructions or a pattern.

Impact

This exhibit will be housed in the Lewisburg Children's Museum for several years and help children learn about rocket launches.

Constraints

The exhibit must be durable, and the game or app must be able to re-set itself so that children do not get error messages when they just press everything at once.

Resources

We have a $2000 budget. We can provide lots of market feedback (i.e., children to test and try to break the software).

Gameplay

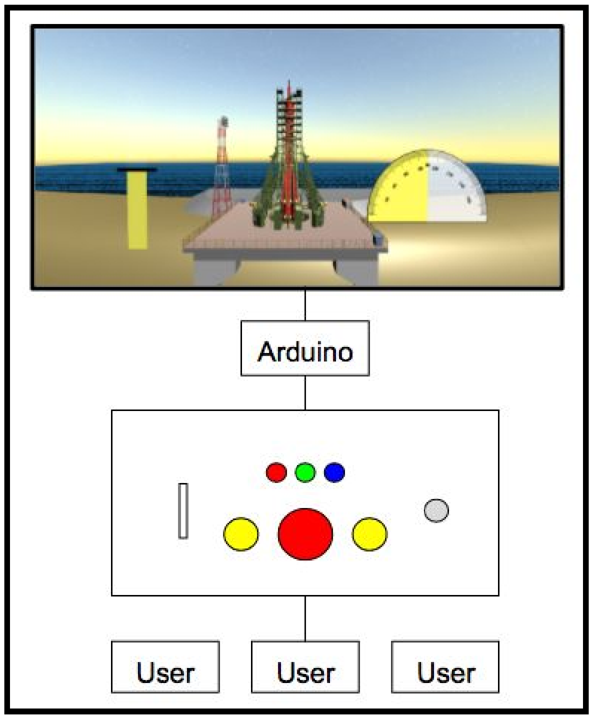


Figure 1: User Gameplay

By examining Figure 1 you can see that there are several users but only one user is directly interacting with the game. This creates a collaborative environment for children to play the game and learn.

While playing the game, the user will be presented with three mission choices. The first mission is launching a satellite into space, the second mission is launching supplies to the International Space Station, and the final mission is launching a rocket to Mars. Each mission involves the user selecting a fuel amount and a trajectory angle to launch the rocket. The main difference between each mission's success is the distance at which the rocket needs to travel into space.

After the user launches the rocket they will get feedback as to whether they successfully launched the rocket an acceptable distance, they launched the rocket too far, or they did not launch the rocket far enough. They can then play the game again and adjust their parameters by learning from their mistakes.

While playing the game there are various sound effects to keep the children engaged. There is also a voiceover for text so that preliterate kids can understand and enjoy the game.

System Design

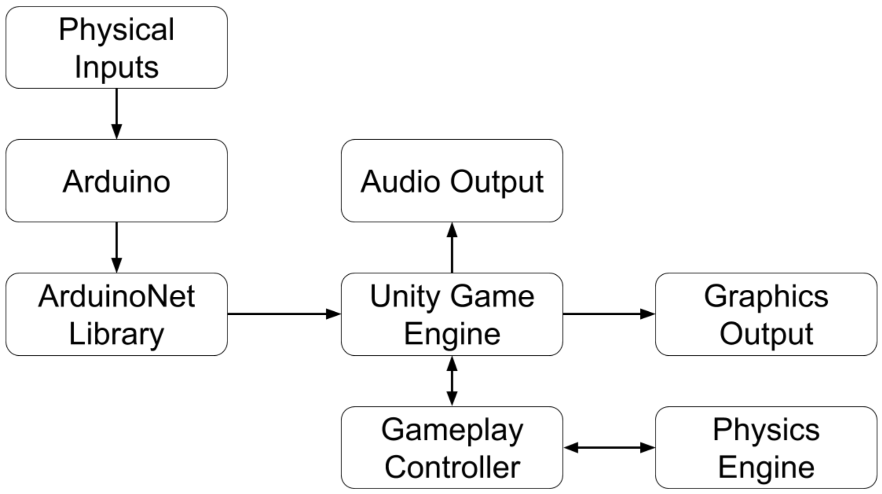


Figure 2: System Design

Figure 2 shows a diagram of our system. Physical inputs coming from the hardware interface are fed into an Arduino microcontroller, where they are processed and sent to the Windows machine via a simple USB connection. Once there, the signals are handled by the ArduinoNet library, which converts changes in these digital signals to events that are compatible with the Unity Game Engine. The Gameplay Controller is a collection of C# scripts that handle the logic of the game, such as navigating the graphical user interface and controlling the graphics of the rocket. The Physics Engine is a special script held within the file RocketBehavior.cs, which applies real-world physics to the rocket. Finally, there is both audio and visual output from our game, which is sent through external speakers and an external display, respectively.

How components are connected together

All the hardware controllers are wired to Arduino Mega, which is loaded with code in the ArduinoNet library. As you can see in the source code, all buttons are configured as INPUT_PULLUP input whereas the slide and knob are treated as analog inputs.

INPUT_PULLUP requires one of the button pins connected to the GND and the other connected to a normal digital Arduino pin. For detailed instructions about how this works, please refer to this manual provided by Arduino. A breadboard is used as there are not enough GND pins for the connection. The connection for the knob and slide is a standard potentiometer connection. Pins should be connected to GND, 5V power source, and analog input pins. In our case, A0, and A1 are used to receive analog inputs.

Although some buttons are LED enabled, we chose not to use LEDs to avoid distracting the kids. However, the can be easily implemented in the future - two side pins on the button are reserved for the LED connection. You should test to see which pin should be connected to the ground.

How Arduino reads the input and sends it the Unity game engine

The Arduino side is implemented as a simple polling method. To debounce the input, the Arduino only reads the input after a short period, rather than each loop cycle. To determine the button's state of being pressed, it needs to remember the previous state. For instance, if the previous button's state is LOW and the current reading is HIGH,then it will signal that a button was pressed. However, if both the previous and current button state are HIGH (user is holding the button), then the button pressed signal will not be fired. The same concept also applies to the analog input. However, since analog inputs can be very unstable, the Arduino needs to set a threshold to determine if the analog input is indeed changed.

To send commands to the Unity game engine, the Arduino employs traditional serial communication protocol, as shown in the Serial Communication Protocol. This protocol documentation specifies all the requirement and implementation suggestions to handle serial communication with the Unity engine.

Where to obtain the hardware components

In the event that the users break components of the hardware controller, such as the slides or buttons, these can be easily replaced. Most of the the hardware components are purchased off-shelf. Here is a completed list of vendor links where we purchased the hardware components:

1. Slide potentiometer: http://www.mouser.com/ds/2/54/ta-778345.pdf.
2. Arcade buttons: https://www.adafruit.com/products/473.
3. LED buttons: https://www.adafruit.com/products/1185 and https://www.adafruit.com/products/1193.
4. Potentiometer: https://www.adafruit.com/products/356
5. Knob: https://www.amazon.com/Uxcell-Aluminum-Potentiometer-Diameter-Knurled/dp/B0173B6K84/ref=sr_1_7?ie=UTF8&qid=1487445310&sr=8-7
6. Arduino Mega: https://www.adafruit.com/product/191

There are several things that need to be manufactured or 3D printed:

1. The knob for the slider has to be 3D printed. The Solidworks file is in docs/SliderKnob.SLDPRT
2. The panel has to be laser cut. The schematic diagram is in docs/Case.dwg

User Interface

The game's user interface can be divided into two components: the hardware interface and the graphical user interface (GUI). Both interfaces have been extensively user-tested to ensure that they are as intuitive to use as possible.

Hardware Interface

The hardware interface is the controller. It consists of six multicolored buttons of various sizes, one slider, and one knob. The hardware components of this interface are laid out radially around the large, red launch button. This is to emphasize spatial significance. The red button in the center is the most important button in the game, acting as both a "select" and a "launch" button, depending on context. Next to it on either side are the yellow arrow buttons. These buttons allow the user to select their mission on the Select Mission Screen. Pressing the left button will shift the selected mission to the left, and likewise selecting the right button will shift the selected mission to the right. Above these three buttons are much smaller colored buttons. These buttons simply change the color of the rocket while it is on-screen. Since the color of the rocket does not impact gameplay in any meaningful way, these buttons are smaller and farther away to reflect that. Finally, the slider to the left and the knob to the right are positioned so as to mirror their in-game counterparts

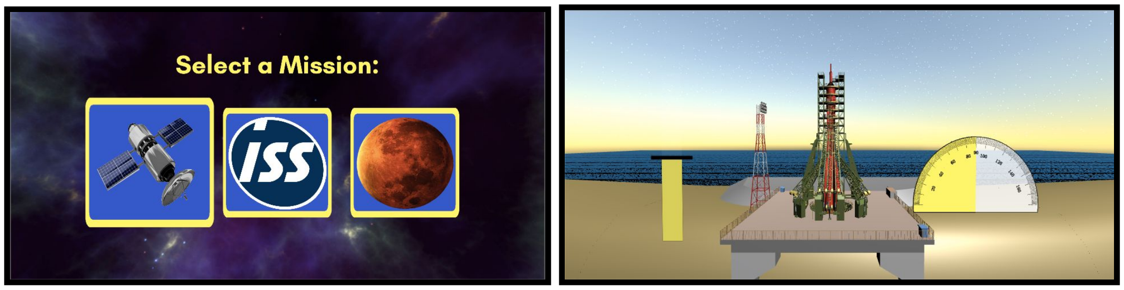
.


Figure 3: The Graphical User Interface (GUI) on the Select Mission Screen and the primary gameplay screen

Graphical User Interface

The GUI is streamlined to be as neat and accessible as possible. Using yellow text and UI elements on a predominantly purple background provides plenty of contrast for those that may have color blindness. Additionally, voiceovers play on most screens to let the user know what they should do or what mission they are attempting. This is mainly to make the game both enjoyable and understandable for preliterate children. At any given point in time, the only UI elements that are present on the screen are text and buttons. Keeping the user interface as clean as possible keeps the children directed and focused on the task at hand.

On the primary gameplay screen, the interface changes. The yellow and black slider on the left matches the fuel that is indicated by the slider on the controller. Likewise, the yellow and gray protractor reflects the angle specified by the angle knob. Providing a one-to-one relation between these respective elements gives the user a direct sense of how their actions affect the game.

Rocket Mechanics

Launching Phase

This is the initial phase where the rocket travels directly upwards at a cubically increasing rate. The rocket will increase its speed until it meets or exceeds a speed equivalent to the value of fuel that was entered. At this point, the rocket will switch to the next phase.
Note: The fuel amount corresponds directly to the velocity.

Turning Phase

During this phase the rocket is traveling at its maximum velocity. The rocket will now slowly turn towards the angle that the user specified. While the rocket turns a vector is created from the rocket's position towards the direction that it is angled. The vector also has a magnitude equivalent to the velocity. The rocket is then moved along that vector. Once the rocket reaches the user's specified angle it will switch to the next phase.

Trajectory Phase

In this phase the rocket has reached the velocity and angle that were specified by the user. Now it will follow a simple trajectory path of an object traveling through space with only gravity from Earth acting on it. This will create a nice parabolic shape.

Equations for the Rocket's Trajectory:
$x_{position} = x_{start} + fuelAmount * cos(\theta) * time$
$y_{position} = y_{start} + fuelAmount * sin(\theta) * time - \frac{1}{2} * 9.8 * time^2$ (Where 9.8 is acceleration due to gravity.)
$z_{position} = z_{start}$

User Stories Overview

Learning Unity

After the team decided that Unity was the platform we were going to use to develop the game we all needed to learn how to use the tool. We did this by assigning various online tutorials for each team member. Each member got two generic tutorials and then a tutorial specific to the part of the project that they were focusing on.

Developing User Interface

The user interface was molded over time through several iterations of rapid prototyping and user testing. The first design in Figure 4 below was a paper mockup that was drawn to reflect how the client envisioned children would interact with the system. It featured a variety of added features, such as a Drop button for dropping the fuel tanks, rocker switches, LEDs, and faceplates. After scrapping some of the more superfluous elements in favor of a cleaner look, this design was user-tested with a group of thirteen kids ranging in age from four to eleven. Based on the valuable data collected during this user test, several major changes were applied to the original design.

The most drastic change to the original design was merging the “Select” and “Launch” buttons into one, and averaging their relative positions on the controller. Since the launch button is the most prominent feature by far, the children wanted to press it constantly. Allowing them to do so without halting the game improved engagement and excitement. Additionally, removing the extra elements and paring down the amount of things the kids could get distracted by improved attention to what was happening on-screen. After scrapping the Drop button in between the second and final iterations and performing another round of user testing, the final controller design in Figure 4 reflects the design that was delivered at the end of the project.

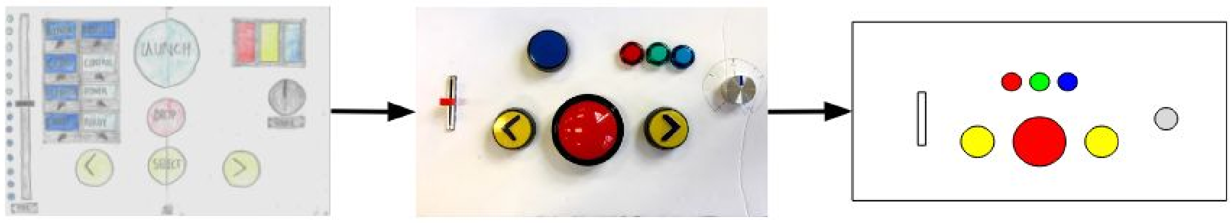


Figure 4: Controller Design Progression

Developing Hardware

We have iterated several sprints to finalize the hardware design. At the beginning we wanted to use a touch screen but decided that hardware components would be more engaging for the kids. The difficult part of developing the hardware was getting the buttons to interact with the arduino’s software and having the arduino send the correct signals to the computer. Due to the Arduino’s user friendly nature, it was relatively simple to wire in the components.

Developing Rocket Mechanics

In order to simulate a rocket launch we sought help from Professor Adam Piggott and Professor Brian Utter.

Professor Piggott is a mathematics professor and he suggested modeling the rocket’s trajectory by creating a free body diagram and calculating the rocket’s position over time. The forces on the diagram would have been gravity and thrust. After developing the diagram we created several equations that would allow me to easily calculate the position within the game. However, after some discussion and contemplation we decided that this approach was useful but it was slightly complex and it might not generate a good path shape.

Professor Utter is a physics professor and he suggested modeling the rocket’s trajectory as the path of throwing a ball into the air where only gravity is acting. All you need in the equation is an initial velocity and a starting location. This approach was very simple and created a nice parabolic shape. For those reasons, this is the approach that we implemented.

Implementing the equations into the game proved to be more challenging than we anticipated. The main challenge had to do with how Unity represents angles. The angle that the rocket is traveling is an important variable in the equations we used to model its trajectory. Unity represents its angles as a quaternion. A quaternion is a complex number instead of just degrees or radians and discovering how to convert between them was difficult. Thus it took time to discover the correct methods to use in the equations.

After implementing the equations we had the trajectory phase, the most difficult phase, complete. The next steps were to implement the launch phase and the turning phase as described earlier.

Integrating the User Interface, Hardware, and Rocket Mechanics
Combining the software and hardware proved to be a challenge. Since the entirety of the physics engine is contained with the file RocketBehavior.cs, it was trivial to add the engine to the work already completed on the user interface. Connecting these two modules to the hardware interface was harder than expected because of how Unity handles external inputs. The serial connection required by the Arduino controller was not directly supported, and had to be completed through a custom library called ArduinoNet. ArduinoNet polls for changes in the signal coming from the microcontroller and converts them into events that can be listened for in software. Once the hardware signals were converted to events, the C# scripts that drive the GUI could perform the appropriate actions based on the kinds of events they received. This completed the bridge between software and hardware.

Implementing Physical Components for the Game

3D Printing
Eric Chesek, an electrical engineering student, aided us in 3D printing a handle for the potentiometer slider. He created a model that we slightly sized down and loaded into the printer.

Laser Cutting
Aaron Clark in the Mechanical Engineering Lab assisted us in laser cutting a box out of acrylic to hold the hardware components. We presented him with a schematic of the design for the top panel (i.e. measurements for where the buttons and potentiometers should be placed). He then took precise measurements of the sizes of the hardware and created a design using online software to create an AutoCAD file. After we purchased the acrylic and he uploaded the design to the laser cutter, the box was quickly cut out.

Final Podium
After our project ends the Lewisburg Children's Museum is going to construct a podium that our hardware and box will be integrated into. Children will be able to sit at the podium in the museum's space exhibit and play the game.

Design Process - SCRUM

We used Yodiz to manage our SCRUM development process. We have 12 sprints and 2 releases. These sprints covered most of the user stories we had at the beginning. The burndown chart for the second release (1/18-5/1) is shown below.

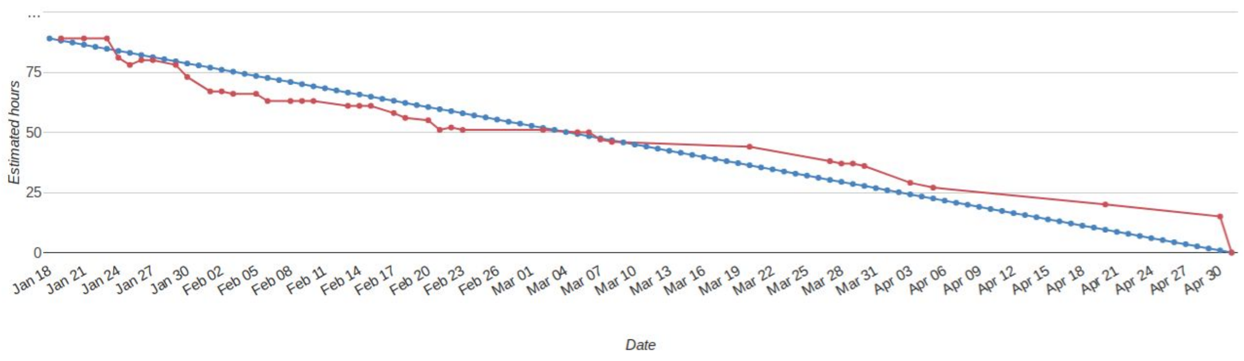


Figure 5: Burndown chart for the second release.

A loose timeline for our SCRUM experience is as follows:

1. Exploring different design options: iPad touch screen interface, Raspberry Pi-driven game, Windows computer and Arduino approach, etc.
2. Learning the Unity game engine through online tutorials
3. Developing features of the game separately: Rocket Mechanics (Michael), User Interface (Kenny), Hardware (Keyi).
4. Integrating all of the separate parts into one product.
5. Finalizing the project: buying materials and constructing the final product.

Unfinished Work

Dropping Fuel Pods
While working on developing the rocket's trajectory, we started implementing the ability for users to drop fuel pods after they run out of fuel. Ideally, this feature was meant to provide something for the children to do while the rocket launches in order to keep them engaged, in addition to adding an entirely new layer of gameplay.

During the game the user would be told to drop a fuel pod at certain points. If they dropped the fuel pod too early or too late then their path would be altered so that they would not successfully complete the mission. Currently there is some code for this that we have commented out. Future teams can use it if they wish to implement this feature.

Mission Feedback
At the end of each mission the users are told whether or not they successfully completed the mission. If they were not successful then they are told whether they went too high or too low, however they are not told specifically why this happened. If we had more time we would have liked to provide the user with information such as their angle being too low or having too little fuel.

Fun Hardware Components
Right now the hardware controller only supports components that are essential for interacting with the game. In order to keep children engaged it could be beneficial to have buttons and lights on the controller that do interesting things, though this should be thoroughly tested in user testing to make sure that these new components aren’t too distracting.

Appendix

Serial Communication Protocol

Overview

This protocol describes a simple protocol used between the Arduino and a computer. It assumes that the host computer has the Arduino driver installed. The protocol supports bidirectional communication. As a result, the host computer is able to send commands to the Arduino, such as turning on the LED. An upstreaming packet (the datagram sent between an Arduino and the host computer) is defined as data sent from the Arduino to the host computer, and vice versa for the downstream packet. Both of the upstream and downstream packet share the same format.

Format

To make it simple, a packet has only two bytes. The first byte indicates the command type, and second byte indicates the command value. The available values for packet type are shown below:

Type Name	Value
NULL	0
BUTTON	1
SLIDE	2
KNOB	3
LED	4

The the second byte (value attribute) depends on what type the packet is:
BUTTON: The value indicates which button has been pressed. It is typically an index value
determined by the wiring order.
SLIDE and KNOB: The value indicates the voltage reading digitized in a 0-255 range.
LED: This is currently unsupported as flashing is not used in the game anymore.

Implementation

The Arduino needs to constantly pull the readings from each individual pin and decide whether to send a message or not. If the Arduino decides that a button is being pressed, it needs to send the message as soon as possible to reduce the latency. Given the loop() execution rate, it is

possible to have multiple inputs sending messages in a small enough time frame that it may appear to update the value simultaneously in the game.
Here are some suggestions for whoever wants to implement this protocol:

- You need to add hardware debouncing, which can be done easily by lowering the sampling rate.
- It is recommended to use event driven design in a different thread. You can check out the ArduinoNet implementation in C# to see how it works.
- To fire the event, it is recommend to keep a previous state so that it won’t continuously send commands when the user is holding the button, for instance.
- For PC-to-Arduino communication, you can implement a message queue in a different thread so that sending a command won’t block the current thread. The queue can be consumed by another thread. This is a typically producer-consumer paradigm.