Introduction

InterROCKtive Curlingization aims to be an educational tool that teaches the rules of curling, as well as a tool for more experienced curlers to simulate different shots.

Motivation

David has been a part of the RPI Curling Club for the past 4 years and has gained significant experience and knowledge of the sport. Eric has curled a couple of times and has a higher than average understanding of the game. As an educational tool, this visualization was motivated by the number of times David has had to explain how curling works to curious parties. As a simulation, the visualization was intriguing because of the ability to construct different scenarios and see the outcome. In games of curling, there are multifarious situations where difficult decisions must be made between which shots to call. These situations often leave players with a burning question: what if? What would have happened if we had thrown the other shot? This visualization provides a method to answer that question via simulating the trajectories of rocks curling and colliding.

Audience

InterROCKtive Curlingization is aimed towards anyone who has seen or heard of curling in passing and would like to learn more about it, as well as curlers of every level. For those that are not well versed in the sport and beginner curlers, the visualization can serve as an educational tool, explaining the rules and providing exposure to the lingo of the game. For the entire audience, the simulation mode of the visualization can be useful. Beginners can simulate shots to gain a better understanding of the mechanics behind curling, while more experienced curlers can use the tool to replay shots from games they may have played and see what would have happened if a different decision was made. The simulation is also simply entertaining for all parties.

Background

Curling Overview

In a game of curling, two teams of four players for eight to ten ends. Club or amateur games are likely to be played for eight ends while professional games will be played for the full ten ends. The four members of each team each have a role:
In each end, each player throws two rocks, each weighing approximately 20 kilograms, in the order described above. The team that has hammer throws the last rock in each end. The two players who did not throw and are not standing on the far side sweep the rock. Sweeping the rock influences how far it goes and how much it curls, or moves from side to side by reducing friction between the rock and the ice and clearing debris from the ice. Vigorous sweeping of rocks can increase the distance they travel by up to a few meters. Sweeping cannot slow rocks down. When each team has thrown eight rocks, the end is over and the team with a rock closest to the center of the house, which is the target on the ice, scores points according to how many of their rocks are closer to the center than the opposing team’s rocks. The team that does not score receives hammer for the following end.

Related Work

Fumito et. al created a real time curling scoring system with the goal of visualizing scores as well as collecting shot accuracy information in “Toward Curling Informatics.” Their work provided good examples for representation of the house and scoreboard. They also provide a very thorough discussion of different factors players consider throughout a curling match and how to represent or account for those visually. The tool Fumito et. al developed is useful for coaching teams and statistical analysis of gameplay. In this capacity, it functions as an educational tool, but only in the hands of someone with enough knowledge of the game to apply the information it provides. [1]

For physical simulation, there are different theories as to what physical phenomena cause a rock to curl. There seems to be a form of consensus regarding a pivoting factor due to an adhesive factor of friction with ice. Shefleki and Losowski briefly summarize and explain the shortcomings of previous theories and propose a model based on the “pivoting-like action” [2]
proposed by Penner, and referencing experimental data on the trajectory of curling in different conditions [3] to confirm the accuracy of their model. The model “couples pivoting with sliding in a quantitative manner” [4] to show that the curl does not increase with the angular velocity of the rock, as Penner stated. [2]

Objectives

There were two main objectives in this project, each with a set of subgoals/constraints:

1. Illustrate how scoring in curling works to a user unfamiliar with the sport
   a. Allow the user to change the gamestate they are visualizing easily
   b. Visualize the current score in real time
   c. Illustrate the meanings of curling terms

2. Visualize curling rock movement
   a. Simulate and visualize the unique curl of the rock when sliding over a large distance
   b. Simulate the collisions of rocks precisely
   c. Allow user to adjust inputs to simulation parameters intuitively
   d. Give instant feedback for changes to simulation parameters in output.

These goals guided the design decisions throughout the project.

Implementation

The main tools used to create this visualization include d3.js and Bootstrap for visualization and UI elements respectively and Python for the server providing simulation results.

Tutorial Mode

This mode of the visualization is meant to function as a learning tool, explaining the rules of curling as well as common terminology used in the game. The UI for this mode is shown below in Figure 1.
Figure 1: Tutorial Mode UI, (a) Controls for visualization, (b) House view with draggable rocks, (c) Scoreboards display live updates as rocks are moved, (d) Instructions for how to use the visualization and general curling info

The controls shown in Figure 1(a) allow the user to reset the rocks to their original position along the top of the house display, to reset the scoreboards, and to record the current score on the scoreboards, advancing the scoreboards to the next end. When the score is recorded any rocks that did not count toward the score for that end are greyed out for clarity. Additionally, the show circles checkbox toggles the measure circles in the house view. This toggle is especially useful if there are many rocks clustered inside the house close together in order to make the visualization more clear as shown in Figure 2. Finally, the switch modes button switches the visualization into simulation mode which utilizes the time slider.

Figure 2: Toggling the measure circles off can significantly reduce visual clutter
The house view is shown in Figure 1(b). In this view there are 16 rocks, 8 for each team. Each rock can be dragged around the entire view. When any part of a rock comes inside the house, a measure circle will appear tangent to the point on the rock closest to the center of the house. The radius of this measure circle will increase and decrease as the rock is moved around, disappearing when the rock covers the center of the house or when it leaves the house. When a rock is outside the house, or inside the house and further away from the center than any rocks from the opposing team, the rock and its measure circle will be darkened. This visual cue is to help differentiate rocks that are sitting (closer to the center of the house than any opposing rocks) from rocks that are not sitting. An example of this is shown in Figure 3. The darkened yellow rock is not sitting because it is further from the center of the house than the red rock. Rocks that are out of play are not darkened. A rock is considered to be in play as soon as it has completely passed the hog line (the thick line at the bottom of the house view) until it completely crosses the backline (the line tangent to the back of the house). This is demonstrated in Figure 4. All of these features accurately represent curling and satisfy objective 1a.

Figure 4: Rocks in play, (a) The darkened rock has not fully crossed the backline so it is in play, (b) The darkened rock has fully crossed the hogline so it is in play
There are two types of scoreboards commonly used in curling, both of which are shown in figure 1(c). The “TV” scoreboard, which is mostly used in televised games or professional matches, functions similarly to a normal scoreboard, displaying how many points each team scores in each end with the total score for each team in the far right column. The “club” scoreboard, which is found in most curling clubs, shows scores along the middle row. The cumulative score for each team is tracked by placing the end number number on the scoreboard. For blank ends (ends where no points are scored) the end number is placed to the far right. Figure 5 shows both styles of scoreboards displaying the score for the same game after six ends. Figure 5(a) shows the club scoreboard. Yellow scored 2 points in the first end, so the end marker with the number one is placed in the 2 column. The second end was blank, so the end marker was placed to the far right. In this visualization, blank ends have a grey background for clarity. In the third end red scored 1 point, then in the fourth end red scored 3 points resulting in 4 total. Thus, the fourth end marker is placed in the fourth column. The fifth and six ends follow the same pattern, with the team’s total score shown by the end marker from the last end they scored in. Through moving rocks around and seeing how the scores change, users are able to interactively learn how the scoreboards work, satisfying objective 1b.

The instructions for how to use the visualization and some general curling information is shown in Figure 1(d). The bold colored text in this section references parts of the simulation or curling terminology. When the user hovers their mouse over any of this text, the corresponding section of the visualization is highlighted. Definitions of curling terms are also given which satisfies objective 1c.

Evolution of Tutorial Mode

Throughout development of this project, no outside feedback was gathered, however both team members critiqued elements of the other’s design. When first designing the house view, the colors of the red and blue rings were very saturated, so much so that it was difficult to see the rocks and measure circles inside the house. Figure 6 shows what this looked like. The colors were changed to have transparency, which, against the white background, actually mimicked the look of a painted house under ice in real life.
The measure circles were still monochromatic at this point and were difficult to see at times, such as when yellow measure circles were in the white section of the house. To fix this, a black circle with slightly larger line width was placed underneath each measure circle so that there was always adequate contrast.

As development continued to be added in order to make the visualization more accurate to an actual curling sheet. These additions included the backline, tee line, four foot lines, center line, and hog line. Eventually, due to the desire to be as accurate as possible the pin, which is a tiny divot at the center of the house, was added as well as a barely visible circle.

When the scoreboards were added, managing the layouts of the divs in html became challenging, so Bootstrap was added. This addition also allowed for prettier buttons. The layout for other non-button controls was then implemented to look similar to the buttons to maintain visual consistency.

A later edition to the UI was rounded corners for the scoreboards and instruction views. This change further added to visual consistency with the controls.

Some future visual changes include splitting up the instructions and general curling info, making each minimizable, and adding a scroll bar so that the entire screen does not need to be scrolled for the visualization to fit.
Simulation Mode

Feedback from the initial proposal recommended that the project focus on visualizing the trajectory instead of the fidelity of the underlying physics being simulated. The first step was to design an architecture for the communication between the frontend and backend. The final architecture is shown in Figure 7.

The simulation is performed on the server for a few reasons. Firstly, this provided a natural encapsulation to the project which allowed the visualization to be designed simultaneously with the simulation. This encapsulation also allows the implementation of the server to change as long as it meets the same API requirements in the future. Secondly, this allows the simulation code to be written in a language other than javascript which is more appropriate for performing mathematical simulation. In this case, Python 3.8 so that the numpy package could be used. Additionally, Python has a culture of easy and quick to get going. Simple tasks take fewer lines. As a result, the code for the server communication portion of the code is 35 lines, importing 5 different 3rd party modules.

Simulation

The simulation itself consists of two components. An implementation of a curling model and an implementation of a collision model.
Normally, a simulation is implemented as a loop of incremental physics calculations. For instance, the straight-forward approach to a simulation for curling would be something like the following pseudo-code:

```plaintext
simulate(rocks):
    dt = .1s
    while any rocks have positive velocity:
        for each rock r:
            r.position += r.velocity * dt
            r.velocity += friction_deceleration * dt
            r.velocity += curling_effect(r.velocity) * dt
        for each pair of rocks (a, b):
            if |a.position - b.position| < 2 * rock_radius:
                a.velocity, b.velocity = collide(a, b)
    output(rocks)
```

This has some drawbacks, however. The precision of the simulation is inversely proportional to the time it takes to run. This is because it needs to run this loop for every time quantum until every rock is at rest. The larger this time quantum is, the fewer loops will run, however, the physics will be more approximate. This is an issue primarily with the collisions, as small errors in the positions of the rocks when they collide result in noticeably erroneous trajectory angles. The trajectory angles need to be precise for this visualization because often in curling rocks follow a chain of collisions, and whether or not these collisions occur is dependent on the angles of the previous collisions.

In order to have a precise yet speedy simulation, a different approach was taken. By using a sufficiently simple model for the motion of a sliding curling rock, the position, velocity, and heading of the rock can be determined as a numerical function of time. This allows the simulation to efficiently search these functions for collision points rather than walk at tiny intervals one at a time.

Such a model was presented by Shegelski and Lozowski [4] which gives a function for the position, speed, and heading of a curling rock as a polynomial function of time. This comes at the cost of some key approximations, however. The main approximation made is the reduction of the rock’s curl to a single constant. This constant is experimentally determined based on the final position of the rock only. This results in the curve of the rock in this model to be realistic to a real curling rock. A real curling rocks “curls” more when it’s slowing down at the end, and makes the curve more pronounced once it passes the hogline. The difference between these is shown in Figure 8.
Shegelski’s Model

Realistic curl

Figure 8: Difference between the simulation model’s curling motion and what a real rock would do. Note the same starting point and angle, as well as the same ending point.

For the purposes of visualization, however, the inaccuracies of this model don’t affect the goals too much, which are more to show the effects of the end of the trajectory, and the game-ramifications of slight changes in collisions.

The algorithm was changed to utilize the advantages of this simplified model and is as follows.
simulate(still_rocks, moving_rocks):
    loop:
        next_t = []
        for each pair of rocks (a in moving_rocks, b in still_rocks):
            next_t.append((time_to_collision(a, b), a, b))
        if next_t.empty():
            break
        t, a, b = min(next_t)
        still_rocks.remove(b)
        a, b = collide(t, a, b)
        moving_rocks.append(a)
        moving_rocks.update(b)
    return moving_rocks

In this version, the outer loop only runs for as many collisions that occur. One notable difference is that in this version it is assumed that no two moving rocks ever collide. It is very difficult to create a realistic situation in curling where two rocks which are moving collide into one another.

The main bottleneck in this version of the algorithm is the search for the next collision in the loop containing the “time_to_collision” routine. This routine works by performing a binary search on the position function to find an upper and lower bound on the time period the moving rock is intersecting the still rock. Then, the binary search is done on these bounds to find the point at which the moving rock starts to intersect the still rock (when their distance is equal to two times their radius).

There’s a global constant which determines how many iterations these searches should run until the value it selects is “good enough.” In the implementation this was chosen as 50, but could probably be lower without any noticeable cost. This is the equivalent of doing the incremental simulation with $2^{50}$ iterations, making it a very powerful optimization.

The collide function is very simple. All it does is find the tangential and normal component of the moving rock’s velocity with respect to its displacement with the still rock. The rocks are assumed to be the same mass, so the momentum transfer results in the moving rock’s normal velocity going to 0, while retaining its full tangential velocity, while the still rock gains the full normal velocity of the moving rock.

The simulation does not account for any energy loss during collisions, as in most cases, this slight loss of energy will only affect the distance and not the angle, and the team found that no loss of energy made it fun to create simulations consisting of many many rocks, which would not happen in a real game.

Use of numpy made these geometric vector calculations very straightforward and easy to read and reason about.
Server/Client Communication

The simulation was encapsulated in the python program in a function which simply took the input parameters (rock positions and velocities), and outputs a sampling of the rocks’ state over their entire trajectory. This was formatted as simply each rock having an array of objects consisting of a time point, position, speed, and heading. Waypoints are generated at set time intervals, the reason for which will be explained in the next section.

The server and client communicate over a protocol known as json rpc (remote procedure calls). Libraries for this protocol handle the serialization of javascript and python objects into JSON which is then sent to and received from the server. Additionally, this protocol allows for registering “methods” which can then be called by the client. In Python this is as simple as adding a single decorator to a function which then allows the server code to dispatch to it with deserialized arguments. This was chosen over a standard REST API as it allows for faster development since the interface is implicit in the registered function signature.

Initially, the server and client communicated over this protocol using HTTP Requests, however it was quickly found that these were rather sluggish and low-throughput. The team switched to using web sockets and found that the latency and throughput were improved greatly, giving the instant feedback that was desired from objective 2d.

Trajectory Visualization

Trajectories were visualized by rendering a line which connects all of the waypoints to show a continuous path, as well as chevrons at each waypoint representing the speed and direction of the rock at those points. The chevron size is proportional to the speed, while its direction was made so that it points in the direction the rock is moving at that time. An example of this is shown in Figure 9.

As the user adjusts the input parameters to the toss, all of the lines and chevrons update instantly to show the result. The visualization elements are animated in d3 so that slow server responses still result in a smooth movement of the trajectories.

In addition to the lines are “preview rocks,” which are transparent versions of the rocks which show their position at a given time. The user can adjust the time slider and all of the preview rocks move to the location they would be along their trajectory at that time. This doesn’t require regenerating the trajectory, so no server call is made in this interaction and it’s very smooth as a result. The position of the rocks are linearly interpolated between their nearest two waypoints. The measure circles are also drawn for these preview rocks instead of the original static rocks so that the user can see how their shot affects the score, as this is one of the primary objectives of this visualization.
Figure 9 - Trajectory Visualization. Each rock has its own color to help distinguish them. Upon close inspection, the blue trajectory moves very slightly to the left after collision, which can be seen by the small chevrons pointing left underneath its yellow ghost rock.

Simulation Controls

The control scheme for the simulation mode of the visualization is designed so that the user has intuitive control over the trajectory of the thrown rock. The inputs the user can change during simulation mode are the speed of the rock (known as “weight” in the sport), the initial angle of the rock (referred to as heading), and the direction of spin of the rock (referred to as
curl). The simulation controls are shown in Figure 10.

As previously mentioned, only the direction of the spin affects the motion of the rock, so the user simply has a slider with two positions to change this value between left and right curls.

The weight is a vertical slider. This is done so that it can be longer overall, giving the user more precise control, and also because it's more intuitive, as the higher the slider is moved, the further the rock will travel up the screen as a result of having more speed at the start.

The method of controlling the angle is the most involved, as this is the most important and precise control a real curler has. In the sport, the thrower attempts to aim at the spot their skip is pointing to at the other end of the ice. The skip is responsible for planning and accounting for the resulting curl of the rock. To mimic this in the visualization, the user may drag a crosshair around the ice indicating where the “skip” is pointing, and therefore, the initial angle of the throw from the other end of the ice.

This effect is further visualized by the “Throw POV,” which shows the side of the ice the thrower is on, and shows an arrow representing the angle of their throw. Very large movements of the crosshair result in large movements of the trajectory, but only slight movements of the arrow representing the actual angle. This was found to give a strong sense to the user how precise the angle is. Additionally, the offset of the trajectory from the crosshair gives the user a sense for how much the rock curled across the entire ice, since the middle 75% of the trajectory can’t be shown due to the awkward aspect ratio of curling ice.

Finally, the time slider is simply a long horizontal slider. This was chosen because time moving from left to right is the general convention used and is instantly understood by users. Behind the scenes, the domain of this slider actually changes depending on how long the
simulation would take. The furthest right point of the slider represents the last millisecond rocks were moving before all coming to rest. If the user changes the simulation parameters and this end-time changes while the slider is at the end, it will stay there, so that the preview rocks are always at their final resting positions. If the parameters are changed and the slider is NOT at the end of the time of the simulation, it will keep its real value, resulting in the preview rocks moving to be at the same time point across the changing simulation. This way if a user is more interested to see how far the rocks get in a certain amount of time with different parameters, they don’t have to constantly move the time slider every time they change any other parameter.

**Conclusion**

This project was a great success overall. Nearly all of the objectives were met, and a very full-featured tool was developed.

The tool adequately illustrates how the current score of an end is calculated, while teaching the user curling terminology. The tool’s simulation mode also gives a novice user a sense of the precision involved in curling, and could even be used by an experienced curler to try and plan a shot in a real match. The tool was even more responsive than the team was aiming for, and this paid off as it made the tool more fun to use.

The only shortcoming of the implementation is that the trajectory isn’t entirely realistic in the shape of the path, so the user doesn’t gain a realistic understanding of how a rock curls. The unrealistic path shape takes away some options that would exist in a real game of curling, such as throwing a rock such that it ends entirely behind another one.

**Future Work**

As of now, the tutorial covers some of the basic terminology of curling, but not many ruled. In the future, more depth could be added, explaining more advanced terminology such as different kinds of shots or confusing rules like the free guard zone.

Some feedback received after presenting the project noted that the rocks did not curl a lot and had a very "flat" trajectory. This was something both David and Eric noticed during development that was a limitation of the model used to represent the motion of the rocks. In the future, a more accurate model could be found, however such a model might negatively impact the performance of the server.

Another possible feature that would not be difficult to add is a "play" button that would play the trajectory of the rock and any collisions that occur in a timeframe that mimics reality. This would add a feeling of realism to the simulation. In addition to this, sound effects could be added that play when rocks collide with each other.

Finally, a Dimp Viz like implementation could be added where the user could drag a rock along its trajectory instead of using a time slider. This would provide more precise control, especially when there are rocks that are moving very fast.
Distribution of Work

David

David designed most of the UI elements and controls for both modes of the visualization. Most of his work was focused on the tutorial mode and transitions between the modes. He also ensured that the scale of all of the UI components was proportional to that of curling in real life.

Eric

Eric implemented the server and simulation within it, as well as designed and implemented the trajectory visualization in the client.
References


