ONTARIO TECH UNIVERSITY
FACULTY OF SCIENCE, COMPUTER SCIENCE

April 6th, 2024

Simulation and Modelling Final Project: Shootout!

by

Alonzo Gabriel Manaog Aidan Mason-Mondesire

Abstract:

For this project, we decided as a team to create 'Shootout!' which is a basketball-like simulation game.

This application was created for the Winter 2023 Simulation and Modelling Course's final project. In the simulation you are able to tweak the power and angle of the ball to try and get it in the basket. Using physics and math concepts to accurately depict the usage of a basketball and how it is thrown. We used Newtonian equations and ordinary differential equations to create an accurate simulation.

1 Introduction

In this report, we are presenting a simulation program done in Python. This simulation is a simulation of 2D projectile motion. We are analyzing 2D motion, as have many others, under Newtonian forces. The program we've written utilizes these forces using multiple different differential equations to simulate the act of throwing a basketball. Velocity, gravity, and angle are but a few of the variables used to create this simulation.

Our motivation for this project is simply an interest in the game of basketball, and wanting to make a recreation of the game. Using our knowledge on 2D projectile motion we were able to recreate it.

Our objectives were to:

1. Develop a game-like application simulating a basketball being thrown.
2. Make the application user-friendly and look nice.
3. Implement equations of motion to the simulation.

Throughout the rest of this report, we will convey the methods used to reach our desired product, the way we implemented it, and what our conclusive product was.

2 Methodology

To make this simulation function, we employed a few methods into the project...
Firstly, we needed to model the simulation. What this entails is deriving the equations for launching projectiles. These equations are derived from Newton's laws of motion [1]. Keeping in mind the constant force of gravity acting on the projectile. For reference, Newton's laws of motion:

1. Newton's First Law of Motion states: If a body is at rest or moving, it was stay at rest or moving constantly unless another force acts upon it.
2. Newton's Second Law of Motion states: The rate of change of momentum of

the projectile is proportional to the net force acting on it. Therefore,

$$F = ma$$

3. Newton's Third Law of Motion states: That for any action, there is an equal and opposite reaction.

In our program it implements these laws. The first and third laws are only used implicitly and not directly programmed into the simulation. The second law however was used explicitly in our program.

In the "Simulation" class, we defined "f" below, in which "ax" and "ay" are our x and y components of the rate of change of momentum otherwise known as Newton's Second Law of Motion.

```
def f(self, t, state, arg1, arg2):
        vx = state[2]
        vy = state[3]
        ax = (-arg1 * state[2])/self.mass
        ay = (-arg2)/self.mass
        return [vx, vy, ax, ay]
```

The next step after derivations was to do the numerical integrations. This was to simulate the projectile motion over time. The numerical technique we used was called the Runge-Kutta method (RK4). To do this in the program we did not do it by hand, but by using the ODE (ordinary differential equation) solver from python's scipy library. The solver computes the numerical integrations for us, and allows us to compute the position and velocity of the ball at each time step. Below is how the ODE solver is used in our code:

```
self.solver = ode(self.f)
self.solver.set_integrator('dop853')
self.solver.set_f_params(self.gamma, self.gravity)
self.solver.set_initial_value(self.pos, self.t)
```

The final part of our methods is to handle the collision and visualize the system. To find collision, we implemented collision detection algorithms to find when and where collisions happen, and then decide what to do. In our simulation we have collision with the ground, collision with the backboard, and with the rim of the net.

The way we handle the collision is based off of the ball reaching a certain position. So if the ball reaches the set of points where the backboard is or the net is, it will run the collision code for what happens after the collision. For example in our step command we find collision response as such:

```
angle_at_impact = math.atan(self.pos[2]/self.pos[3])

new_velocity = math.sqrt(self.pos[2] ** 2 +self.pos[3]**2)

new_velocity_x = 0.2 * new_velocity *
math.cos(np.radians(angle_at_impact)+2.355)

state_after_collision = [self.pos[0],self.pos[1],
new_velocity_x,self.pos[3]]
```

Above is an example of the collision response that we used towards the backboard. It calculates what the angle of impact is, the new velocity, the horizontal component of the velocity, and sets the state of the ball after it collides with the backboard. These equations that we sourced [2], calculate how the ball moves and acts after collision on a vertical wall (backboard).

```
angle_at_impact = math.atan(self.pos[2]/self.pos[3])

new_velocity = math.sqrt(self.pos[2] ** 2 +self.pos[3]**2)

new_velocity_y = 0.2 * new_velocity *
math.sin(np.radians(angle_at_impact)+0.785)

state_after_collision = [self.pos[0],self.pos[1],
,self.pos[2],new_velocity_y]
```

Another example of collision response is the code above which is used for the collision response on the rim and the floor. It calculates the angle of impact is, the new velocity, the vertical component of the velocity, and sets the state of the ball after it collides with the floor or rim. These equations that we sourced [2], calculate how the ball moves and acts after collision on a horizontal wall.

The final section of our method of creating this simulation is visualizing everything. We used pygame to visualize our ball and background to show the program. It is

the medium in which we are able to see the ball and the net, it renders the motion on the graphical interface. Below you can see how the program looks.

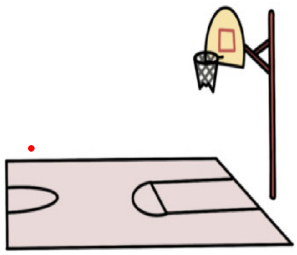

Figure 1: The start state of Shootout!

3 Implementation

There were a few things needed to fully implement this simulation, a few tools were used.

1. Python. A programming language needed to be used to code all of this so python is the approach we took.

2. Dependencies. We needed library dependencies in order to program it. We used numpy and math for equations and math, scipy for solving the ODE's, and pygame for visualizing.

4 Conclusions

In conclusion, we were able to create a basketball-like simulation. We were able to find the correct sorts of mathematical equations and python dependencies to be able to accurately simulate a ball being thrown at a basketball net.

References

1 "Newton's laws of Motion," Encyclopædia Britannica, https://www.britannica.com/science/Newtons-laws-of-motion (accessed Apr. 6, 2024).

2 John Hunter, Angle and velocity of a bouncing ball, URL (version: 2021-06-25):
https://physics.stackexchange.com/q/647757.

A Appendix

- When running the program, make sure to have basketball1.png and the source code python file in the same folder.
- to tweak the values of the simulation, on line 136, the first value is the power and the second value is the angle you are sending it at.