AN ADAPTIVE AND INTERACTIVE 3D SIMULATION PLATFORM FOR PHYSICS EDUCATION USING MACHINE LEARNING AND GAME ENGINE

Weicheng Wang[1] and Yu Sun[2]

[1]Arnold O. Beckman High School, 3588 Bryan Ave, Irvine, CA 92602
[2]California State Polytechnic University, Pomona, CA, 91768

ABSTRACT

When undergraduate students just got into the physics field, it might be difficult for them to understand, think, and imagine what is happening in certain phenomenons [6]. For example, when two objects have different masses and velocity collide into each other, how are they going to act? Are they going to stop, bounce away from each other, or stick together? This simulation helps the students who do not feel comfortable imagining these scenarios.

Currently we only have the gravitation lab, the trajectory lab, and the collision lab. The gravitation lab is a planet orbiting a sun, where the users can input different masses for the sun and planet, and the radius (in AU), the program will then calculate the gravitational force and orbital period while the planet starts orbiting its sun at a certain speed [7]. The trajectory lab is an object doing projectile motion, where the user input variables like initial velocity, angle, height, and acceleration, the program will present current position and velocity on the screen as the object doing projectile motion [8]. The collision lab is where the user input the masses and velocities for the two objects, and after the user decide the collision is going to elastic or not, set the lab and press start, the program will calculate the total momentum and kinetic energy and have it on the right side of the screen while the objects starts colliding [9].

KEYWORDS

Physics, Simulation, Problem Solving, Animation.

1. INTRODUCTION

Recently I have been studying physics, learning different phenomenons, and solving different kinds of questions. However, I do sometimes find out that some phenomenons, like a planet orbiting its sun or when two different objects colliding into each other either in elastic or inelastic conditions, are very hard to imagine or even think of. Therefore, we developed a program to demonstrate how an object, or objects, looks in different situations.

For example, we first applied our application to a situation where you set up the angle, initial velocity, acceleration, and the height of an object you want to start off. Then, we conducted a qualitative evaluation of the approach; the results came out the same as how we calculated it, and it fully demonstrates what will happen after a period of time.

David C. Wyld et al. (Eds): COMIT, CRBL, BIOM, WiMNeT, SIP, AISO - 2021
pp. 47-62, 2021. CS & IT - CSCP 2021 DOI: 10.5121/csit.2021.111705

Some of the techniques and systems like “myPhysicsLab” by Erik Neumann that have been proposed to demonstrate how objects work, which also allows the user to change the conditions of labs, to test out and see how objects act in different conditions [15]. However, the proposal assumes people who intended to use this to know variable names that are not commonly known or used in physics like damping [2]. It is also hard for people who just got into physics to use, because normally we learn kinematics, which is associated with projectile motions, and this proposal is mainly focused on wave topics, which won’t be taught until nearly the end of the course.

Other physics simulations, such as the Cliff Diver in exploration series by the CK-12 Foundation, are using simple terms and have very nice graphics [14]. However, it has barely any variables that can be changed [3]. They are more like a game instead of a lab, which is what I wanted.

I used unity as a tool and C# as a coding language to make this, and it’s mainly because Unity is very effective while rendering 2D and 3D scenes, and the quality offered is also relatively good compared to other apps [10]. The reason I used C# is because it is somewhat like java, but faster. The program I created is more of a tool that helps people to understand physics better, to demonstrate and help people with physics problems.

There are three labs I have created so far, the gravitation lab, the trajectory lab, and the collision lab. The gravitation lab is a planet orbiting its sun, the user can change the masses of the sun and the planet, and the radius between them, and after they press “set” and “play”, the planet starts orbiting. It also shows some outputs like the current gravitational force and the current orbital period. The trajectory lab demonstrates the projectile motions, the user can set the conditions like angle, the height, the initial velocity and acceleration of the object. After clicking “set”, it will predict the path the object will go through, with outputs like current position and current velocity. You can also change the time interval, so that it goes by that time interval every time you click “step”. The collision lab is going to be a lab with two objects inside, you can set the mass and velocity of them, and on the right side it will show the total kinetic energy and momentum. You can also set the condition to elastic or perfectly inelastic. Compared to some other methods, this tool is surely easier to use for those beginners who just got into physics, and also more like a lab instead of a game.

In this paper, we follow the same line of research by solving different physics problems. Our goal is to make sure it’s correct and shows exactly what is going on. Our method is inspired by physicslab, and there are some good features, such as clear and easier to understand. Therefore, we believe that it helps people to get into physics much easier.

To prove that the labs are evaluating the results correctly, I first randomly generated twenty lists of numbers, calculated them by hand, and inputted them into the labs. The outputs were very accurate, so were the demonstrations. However, because I used float instead of double, when it comes to some numbers like the square root of two or ⅓, there might be an almost irrelevant difference between. Second, I asked some random students who are either taking, or took physics to fill up the survey. The result came up that most of them think it would be really helpful, especially the gravitation and trajectory lab, because they think it could have saved so much time if they had it.

In two application scenarios, we demonstrate how the above combination of techniques increases ... First, we found some physics problems that had to do with the labs and made sure they all gave the right output we wanted. Second, we let some people test it out and make sure there are no errors or bugs that are unexpected and need to be fixed.

The rest of the paper is organized as follows: Section 2 gives the details on the challenges that we met during the experiment and designing the sample; Section 3 focuses on the details of our solutions corresponding to the challenges that we mentioned in Section 2; Section 4 presents the relevant details about the experiment we did, following by presenting the related work in Section 5. Finally, Section 6 gives the conclusion remarks, as well as pointing out the future work of this project.

2. CHALLENGES

In order to build the tracking system, a few challenges have been identified as follows.

2.1. Self-learning coding

The first challenge I faced was the fact that I am very new to coding, and have only used python and a little java. Therefore I asked my teacher to see if he had any recommendations. He told me that he doesn't recommend me to use python, and provided me with a few other programming languages. After looking over them, I decided to use c#, on unity. Then I started learning how to use c# by watching YouTube videos and googling. At first, it was super hard because it was my first time learning a programming language on my own, but then I got better and better. I also realized that c# is very similar to python and java, especially java: for example, the semicolon you have to put at the end of a line, use parentheses and functions etcetera.

2.2. Equations to methods

The second challenge was the fact I also had to self teach some physics. The reason behind it was because some of the things (like the period equation in the gravitation lab) we didn't really learn in physics A, and some equations needed to be changed so I can put them into code, and provide a correct output. I solved this problem by countless tries and by reviewing and understanding more about physics. However, there are still some issues like the fact that the collision lab only gives the total momentum and kinetic energy, that was mostly because I didn't have enough time to re- code for the momentum and kinetic energy for each one, I will probably fix it in the future [11].

2.3. Modeling

The third challenge is when I try to combine the code with unity. Unity is harder to use than it looks, it has a lot of features, which is difficult for new users like me to use smoothly. I first tried to explore myself, but then after I turned my files into a disaster and had to delete and re-download unity, I decided to watch the tutorial that teaches you how to use unity "properly". After a list of trial and error, for example forgetting where I put the object and ended up lots of planets crashing into each other, or a block that just won't stop going down, also there's that time I accidentally put in too much stuff so the unity keep on crushing on me, I finally combined the script and models together.

3. SOLUTION

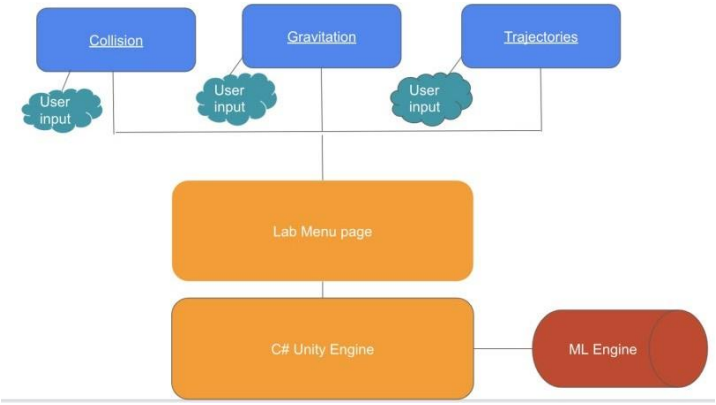

Figure 1. The overview of the project

Phymulation is a program that helps students who find it hard to imagine certain situations in physics to get a better understanding. It's written in C# and run by ML Engine in Unity. When the user first loads in, they will see the menu page, and then they can choose which lab they want to use. For the collision lab, you input the masses and velocities of the two objects, and set it. It will provide the total momentum and total kinetic energy on the right while the objects collide with each other. Then, for the gravitation lab, the user input the masses of the sun and the planet, and then input the radius(in AU), the program will calculate the gravitational force and orbital period, and show it on the screen. The trajectory lab is when the user inputs the initial velocity, angle, height, and acceleration, the program will output the current position and velocity on the screen as the object doing projectile motion.

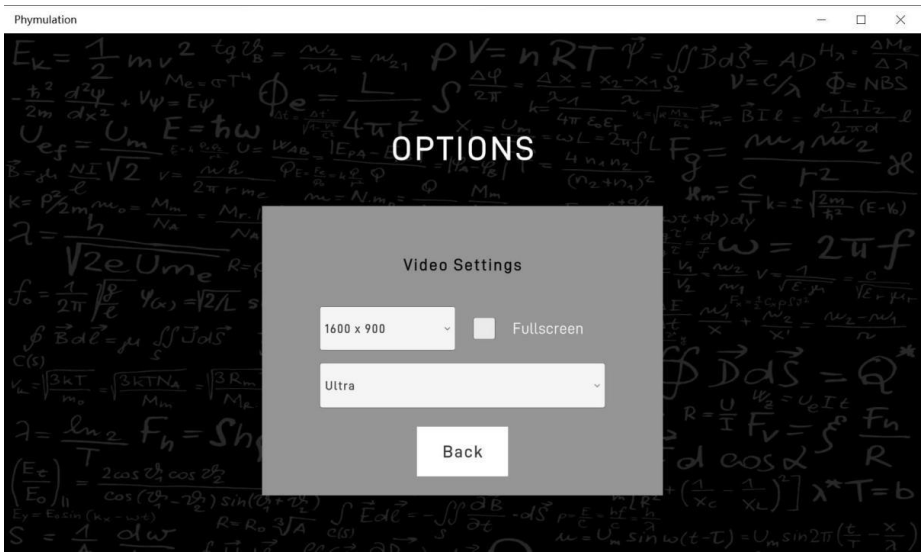

Figure 2. The screenshot of settings.

```
public void Start()
{
    resolutions = Screen.resolutions;
    resolutionDropdown.ClearOptions();
    List<string> options = new List<string>(); for(int i = 0; i < resolutions.Length; i++)
    {
        options.Add(resolutions[i].width + " x " + resolutions[i].height);
        if(resolutions[i].width == Screen.currentResolution.width && resolutions[i].height ==Screen.currentResolution.height)
        {
            resIndex = i;
        }
    }

    resolutionDropdown.AddOptions(options);

    if (PlayerPrefs.HasKey("resValue"))
    {
        resolutionDropdown.value = PlayerPrefs.GetInt("resValue");
        resolutionDropdown.RefreshShownValue();
        resValue = PlayerPrefs.GetInt("resValue");
        resWidth = PlayerPrefs.GetInt("resWidth");
        resHeight = PlayerPrefs.GetInt("resHeight");
    }
    else
    {
        resolutionDropdown.value = resIndex;
        resolutionDropdown.RefreshShownValue();
    }


    if (PlayerPrefs.HasKey("isFullscreen"))
    {
        isFullscreen = PlayerPrefs.GetInt("isFullscreen");    if (PlayerPrefs.GetInt("isFullscreen") == 1)
            fullscreenToggle.isOn = true;else
```

```
        fullscreenToggle.isOn = false;
    }

    if (PlayerPrefs.HasKey("qualityLevel"))
    {
      qualityDropdown.value = PlayerPrefs.GetInt("qualityLevel");
      qualityDropdown.RefreshShownValue(
      );              qualityLevel              =
      PlayerPrefs.GetInt("qualityLevel");
    }


    resolutionDropdown.onValueChanged.AddListener((value) =>
    {
      resWidth                  =
      resolutions[value].width;
      resHeight                 =
      resolutions[value].height;
      Screen.SetResolution(resWidth,            resHeight,
      Screen.fullScreen);resValue = value;
    });

    qualityDropdown.onValueChanged.AddListener((value) =>
    {
      QualitySettings.SetQualityLevel(value)
      ;qualityLevel = value;
    });

    fullscreenToggle.onValueChanged.AddListener((value) =>
    {
      Screen.fullScre
      en = value; if
      (value)
      {
        isFullscreen = 1;
      }
      else
      {
        isFullscreen = 0;
     }
     }
)
;

}
```

This part of the code is to set the quality, resolution, and full-screen if the user wished to. I created a new list “options” to save the preset length and width resolutions, and added them into “drop- down”, so the user can choose from different resolutions like 1920x1440. Then I created a

full- screen toggle so the user can set the simulation into full-screen. I also made a quality drop-down to let the user choose the quality level they would prefer.

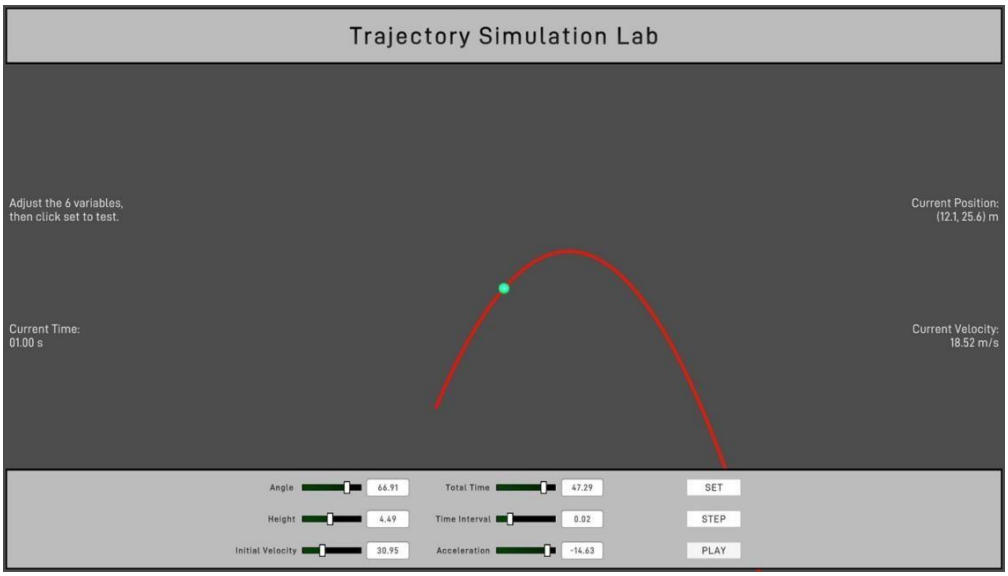

Figure 3. Screenshot of trajectory Simulation Lab

```
    public void SetVariables()
    {
        if(!float.TryParse(angleField.text, out float result) || !float.TryParse(heightField.text, out float result2) || !float.TryParse(velInitField.text, out float result3) || !float.TryParse(accelerationField.text, out float result4) || !float.TryParse(timeIntField.text, out float result5) || !float.TryParse(totalTimeField.text, out float result6))
        {
            return;
        }

        angle = float.Parse(angleField.text);
        height = float.Parse(heightField.text);
        velInit = float.Parse(velInitField.text);
        acceleration = float.Parse(accelerationField.text);
        timeInterval = float.Parse(timeIntField.text);
        totalTime = float.Parse(totalTimeField.text);

        PlayerPrefs.SetFloat("angle", angle);
        PlayerPrefs.SetFloat("height", height);
        PlayerPrefs.SetFloat("velInit", velInit);
        PlayerPrefs.SetFloat("acceleration", acceleration);
        PlayerPrefs.SetFloat("timeInterval", timeInterval);
        PlayerPrefs.SetFloat("totalTime", totalTime);

        segments = (int)(totalTime / timeInterval);
        points = new Vector3[segments];
        arcPoints = new Vector3[segments];
```

```
        float x = 0;
        float y = 0;

        velXInit = velInit * Mathf.Cos(Mathf.Deg2Rad * angle);
        velYInit = velInit * Mathf.Sin(Mathf.Deg2Rad * angle);

        for (int i = 0; i < segments; i++)
        {
            x = velXInit * timeInterval * i;
            y = (velYInit * timeInterval * i) + height + (0.5f * acceleration * Mathf.Pow(timeInterval
* i, 2));
            points[i] = new Vector3(x, y, 0);
            arcPoints[i] = new Vector3(x*0.1f, y*0.1f, 0);
        }

        currentPoint = 0;
        sphere.transform.position = arcPoints[currentPoint];

        lr.positionCount = segments;
        lr.SetPositions(arcPoints);
        SetText();
    }
```

In “setVariables”, I first check if the user input can be turned into a float, if it can, then return it. Then I set the angle, height, initial velocity, acceleration, total time, and time interval from a string (where the user imputed) into floats, and also set up the lab. Next step was to calculate the initial velocity in x and y directions to calculate the current position. The last thing (in shown codes, not the actual code) I did was to count and set the positions, and also printing it out.

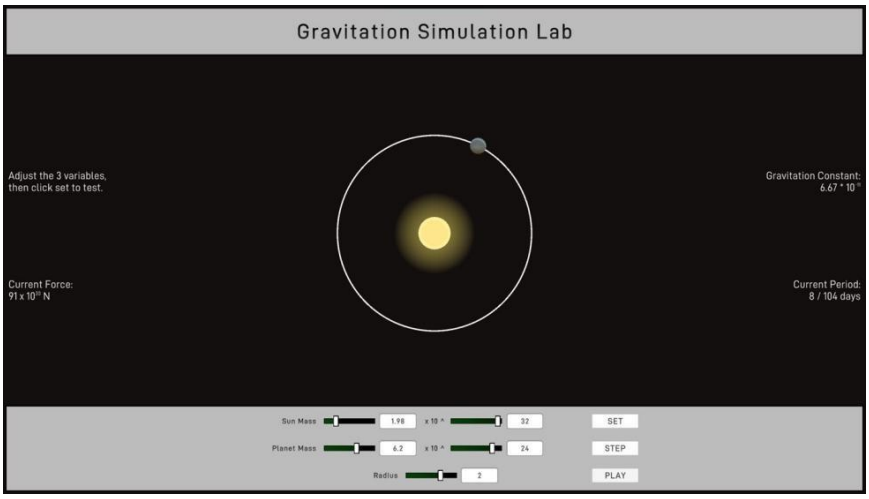

Figure 4. Screenshot of gravitation Simulation Lab

```
void CalculateVariables()
{
    orbitPath = new Circle(radius);
    orbitPeriod = Mathf.Sqrt((4 * Mathf.Pow(Mathf.PI, 2) * Mathf.Pow(radius*AU*Mathf.Pow(10,3), 3)) / (float)(G * massOfCenter * Mathf.Pow(10, massCenterPower)));
    orbitPeriod = (((orbitPeriod / 60f) / 60f) / 24f);
    force = (6.67f * massOfCenter * massOfOrbiting) / Mathf.Pow(radius * 1.49598f, 2);
    forcePower = 0;
    if(force > 10)
    {
        force /= 10;
        forcePower += 1;
    }
    if(force < 10)
    {
        force *= 10;
        forcePower -= 1;
    }
    forcePower += massCenterPower + massOrbitPower - 25;
    timeInterval = orbitPeriod / segments;
}

public void SetText()
{
    currentForceText.text = "Current Force:\n" + Mathf.RoundToInt(force).ToString() + " x 10<sup>" + forcePower.ToString() + "</sup> N";
    currentPeriodText.text = "Current Period:\n" + Mathf.RoundToInt(currentPoint * timeInterval).ToString() + " / " + Mathf.RoundToInt(orbitPeriod).ToString() + " days";
}
```

The orbitPath is to create an orbit based on the radius(in AU) provided. Using that we can calculate the orbit period by using the equation(T=42r3GM), and then convert it into days. Then I used another equation (Fg=GGm1m2r2) to calculate the gravitational force, and then I used a list of concepts, like keep dividing it by 10 until it's less than 10 and add a power to 10 every time, to convert it into scientific notation form. At last I used "Set Text" to set the text that would be printed out.

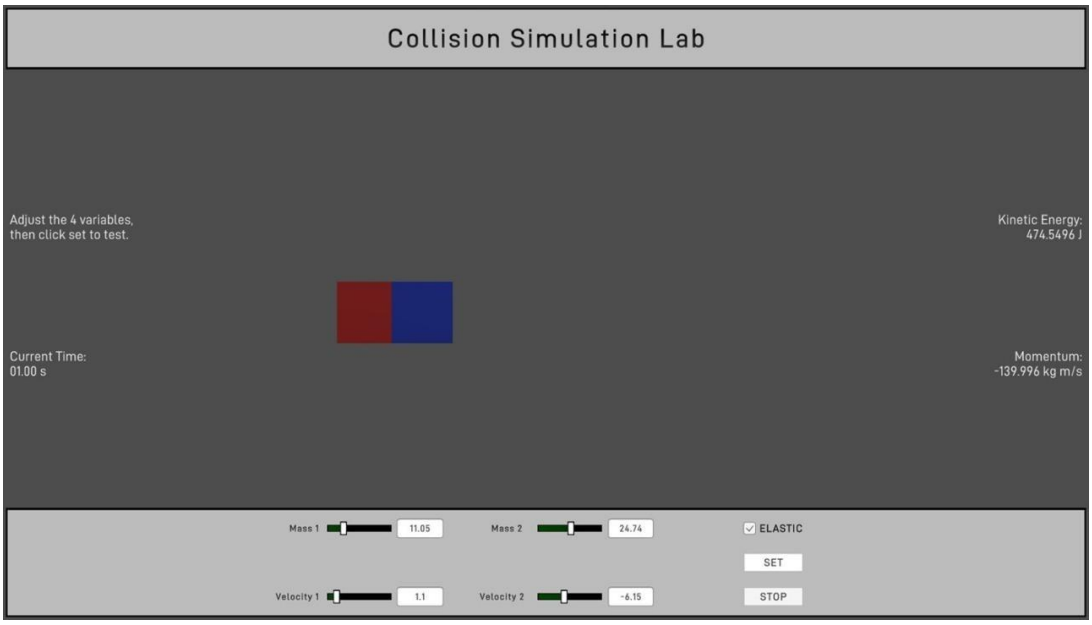

Figure 5. Screenshot of collision Simulation Lab.

```
 public void InitSetVariables()
{
    If(!PlayerPrefs.HasKey("mass1")||!PlayerPrefs.HasKey("mass2")
|| !PlayerPrefs.HasKey("vel1") || !PlayerPrefs.HasKey("vel2") || !PlayerPrefs.HasKey("elastic"))
    {
        return;
    }

    cube1.transform.position = new
    Vector2(-4, 0); cube2.transform.position
    = new Vector2(4, 0); rb1.mass = mass1;
    rb2.mass = mass2;

    if (isElastic)
    {
        material.bounciness = 1f;
    }
    else
    {
        material.bounciness = 0f;
    }

    CalculateVariables(
    );SetText();
}
void CalculateVariables()
{
    kineticEnergy = (0.5f * mass1 * Mathf.Pow(vel1,2)) + (0.5f * mass2 * Mathf.Pow(vel2,
    2));momentum = (mass1*vel1) + (mass2 * vel2);
}
```

```
public void SetVariables()
{
    if (!float.TryParse(mass1Field.text, out float result) || !float.TryParse(mass2Field.text, out float result2) || !float.TryParse(vel1Field.text, out float result3) || !float.TryParse(vel2Field.text, out float result4))
    {
        return;
    }

    mass1 = float.Parse(mass1Field.text);
    mass2 = float.Parse(mass2Field.text);
    vel1 = float.Parse(vel1Field.text);
    vel2 = float.Parse(vel2Field.text);
    isElastic = elasticToggle.isOn;

    PlayerPrefs.SetFloat("mass1", mass1);
    PlayerPrefs.SetFloat("mass2", mass2);
    PlayerPrefs.SetFloat("vel1", vel1);
    PlayerPrefs.SetFloat("vel2", vel2);
    PlayerPrefs.SetString("elastic", isElastic.ToString());

    if (isElastic)
    {
        material.bounciness = 1f;
    }
    else
    {
        material.bounciness = 0f;
    }


    cube1.transform.position = new Vector2(-4, 0); cube2.transform.position = new Vector2(4, 0); rb1.mass = mass1;
    rb2.mass = mass2;

                    CalculateVariables();SetText();
```

First of all, I used initSetVariables to check if the user set any variables like mass and velocity, and check if the user set the system into elastic. Secondly, I made the objects(cubes) transform into positions (4,0) and (-4,0). If the user clicks the set button, the simulation will set the objects' masses exactly like how the user inputted it, and set the system into elastic if the user set it to elastic. The third thing was to calculate the kinetic energy and momentum, then display them. After all that I let the variables, which were set by the user, to show on the screen, and wait for the player to either change them, or start the lab.

4. EXPERIMENT

4.1. Experiment 1

To test if each lab is accurate, I found 20 questions for each lab to test their accuracy. For example, I used the actual value from some solar systems to test out the gravitation lab, they worked out perfectly. However, sometimes(especially when including numbers like square root of 2 etc) the result comes out to be approximate. I think it is because the float only saves values up to 6 to 7 desmos, therefore when facing some numbers that have more than 7 desmos or continues going, it won't be as accurate. The same problem exists in the Trajectory lab. However, unlike the other two labs, the collision lab actually worked more accurately. I think the reason might be that there are not many divisions, especially during the calculation of total momentum(the equation was mass times velocity for individual one, and the total momentum was the sum of those), which only includes mutualisation.

Catagory	Index	Result By Program (in days)	Actucal Result(in days)
Gravation Lab	1	365	365
Gravation Lab	2	88	88
Gravation Lab	3	687	687
Gravation Lab	4	1898	1898
Gravation Lab	5	529	529
Gravation Lab	6	3	3
Gravation Lab	7	2600	2600
Gravation Lab	8	1546	1546
Gravation Lab	9	48878	48878
Gravation Lab	10	154566	154566
Gravation Lab	11	47178012	47178012
Gravation Lab	12	809996608	8099966010
Gravation Lab	13	1545657216	1545657217
Gravation Lab	14	1995434624	1995434625
Gravation Lab	15	384	384
Gravation Lab	16	111	111
Gravation Lab	17	745	745
Gravation Lab	18	2167	2167
Gravation Lab	19	3542	3542
Gravation Lab	20	28597	28597

Table 1. The output, total period, of the gravitation lab

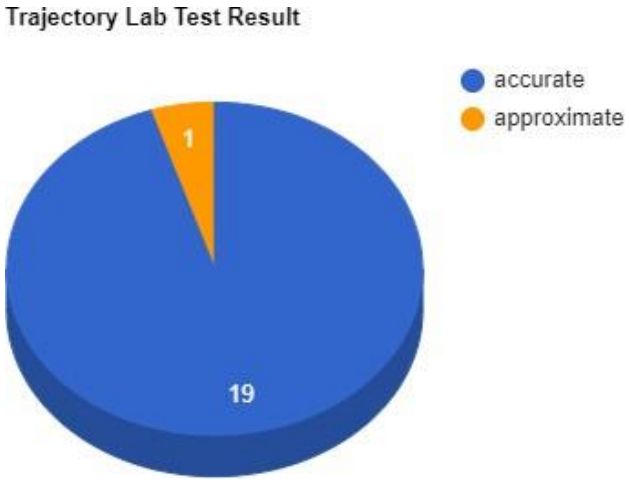

Figure 6. The test result of Trajectory Lab

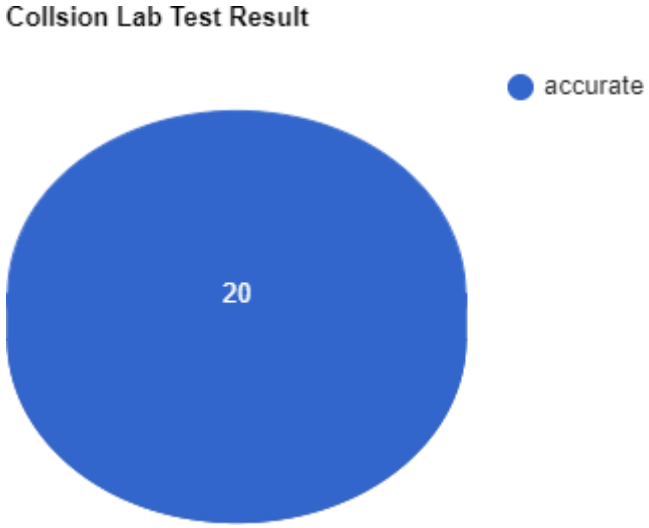

Figure 7. The test result of Collision Lab

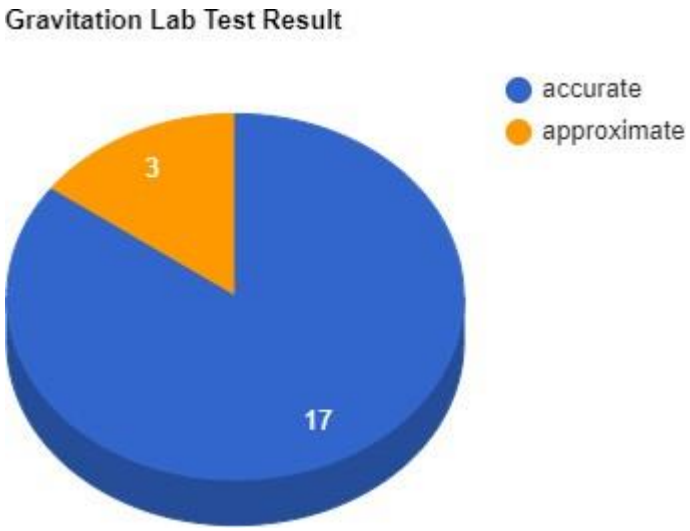

Figure 8. The test result of Gravitation Lab

4.2. Experiment 2

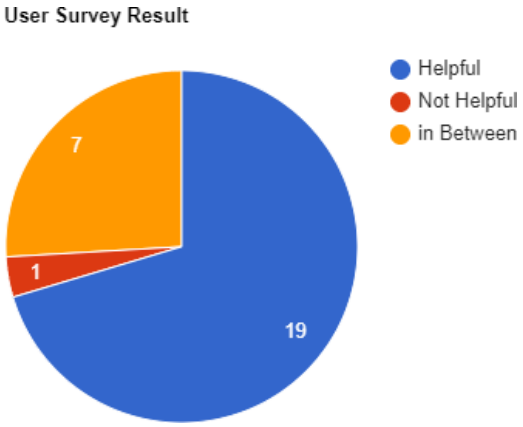

Figure 9. The result of user survey

I presented the program to twenty-seven people that had physics last year, and the result came up above as a pie chart. Most people are saying that it's helpful, especially the gravitation and trajectory lab because they think it could have saved so much time if they had it. The most common reason for those who were saying "in-between" was because it can only calculate certain values in certain conditions. For example, if you had the gravitational force and the radius and wanted to find the mass of the sun, it doesn't really help. The only one that said "not helpful" gave out three reasons: first, the radius of the gravitation lab was in au, and we didn't learn it during school year; secondly, the collision lab on gives the total kinetic energy and total momentum, which is actually very easy to calculate; third reason was that the trajectory lab, you can only do the problems that provide the angle, height, initial velocity and acceleration.

For the first experiment, I calculated and compared the answer between the output where the program gave me and the actual values. Although there were differences between the answers, it only exists if the value is too large, and the differences were not very significant. The most common reason why people think that the program is helpful is because it can only calculate certain outputs with certain inputs. However, the main goal of this program is to let the people understand more about physics, and to help students to imagine certain situations better instead of a calculator that helps you to solve problems. As long as it works similar to what actually happens, it will reach my expectations.

5. RELATED WORK

Chytracek et al. presented a special language to describe the geometries of detectors related with the physics measurement, in order to analyze the solids and materials [1]. Our work focuses on creating a lab and demonstrating how objects act in different conditions using a game engine. Our work does not rely on a modeling language, but uses a 3D game engine for visual simulation.

K. Binder et al. created a simulator for those already familiar with the theory of critical phenomena, but it would be difficult, and mostly impossible for the undergraduate physics students to use [4]. Our work is much simpler, and is good for students who just got into physics to use, to help them get more into physics.

José M. Carcione et al. develop a numerical algorithm for a wave propagation in linear nonisothermal poroelastic media simulation [5]. Our work demonstrates projectile motion, how

planets orbits, and what would happen when two objects with different mass and velocity collide into each other in three different labs.

6. CONCLUSIONS

This is a program for better understanding of physics for students who are undergraduates and are new to the physics field. We used three different labs: Gravitation Lab, Trajectory Lab, and Collision Lab to demonstrate how planets orbit, projectile motions, and how objects collide. To test if the program is efficient, I randomly generalized 20 different lists of inputs, and made sure the output is accurate, and the objects are behaving how they should behave [12]. Then I asked twenty-seven students that had physics last year to finish a survey on whether they think the app is helpful, most of them think that it is helpful, some others said it needs improvement. However, they all admit I have reached my goal, which is to demonstrate how objects behave under certain conditions.

One of the limitations was the accuracy, even though it doesn't make a big difference in between [13]. To solve this, I think I can change the float into double, because the precision of float is only six or seven decimal digits, while double variables have a precision of about 15 digits. The other one was that in the gravitation lab, the planet rotates too slow. I am thinking of adding a "speed up" button so the users can speed it up. For the collision lab, I will add in the kinetic energy and momentum of each object so it could be more helpful. I can also add in other labs like "rotation lab" and "oscillation lab" for people that find it hard to imagine these things. The fact that it only can calculate certain outputs by putting in certain inputs might also be solved, although this wasn't really a part of my goal, but since people wanted this feature, I can probably make it real.

REFERENCES

[1] Chytracek, Radovan, Jeremy McCormick, Witold Pokorski, and Giovanni Santin. "Geometry description markup language for physics simulation and analysis applications." IEEE Transactions on Nuclear Science 53, no. 5 (2006): 2892-2896.

[2] myPhysicsLab https://www.myphysicslab.com/

[3] Cliff. Diver, https://interactives.ck12.org/simulations/physics/cliff-diver/app/index.html?screen=sandbox&lang=en&referrer=ck12Launcher&backUrl=https://interactives.ck12.org/ simulations/physics.html

[4] Binder, Kurt, et al. "Monte Carlo simulation in statistical physics." Computers in Physics 7.2 (1993): 156-157.

[5] Selvadurai, Antony PS, ed. Mechanics of poroelastic media. Vol. 35. Springer Science & Business Media, 1996.

[6] Heisenberg, Werner. Physics and beyond. London: Allen & Unwin, 1971.

[7] Boynton, Paul E., et al. "Gravitation physics at BGPL." New Astronomy Reviews 51.3-4 (2007): 334-340.

[8] Zheng, Yu. "Trajectory data mining: an overview." ACM Transactions on Intelligent Systems and Technology (TIST) 6.3 (2015): 1-41.

[9] Goldberger, Marvin L., and Kenneth M. Watson. Collision theory. Courier Corporation, 2004.

[10] Harman, Philip V., et al. "Rapid 2D-to-3D conversion." Stereoscopic displays and virtual reality systems IX. Vol. 4660. International Society for Optics and Photonics, 2002.

[11] Jegadeesh, Narasimhan, and Sheridan Titman. "Momentum." Annu. Rev. Financ. Econ. 3.1 (2011): 493- 509.

[12] Bushnell, David S. "Input, process, output: A model for evaluating training." Training & Development Journal 44.3 (1990): 41-44.

[13] Diebold, Francis X., and Robert S. Mariano. "Comparing predictive accuracy." Journal of Business & economic statistics 20.1 (2002): 134-144.

[14] Christensen, Neil, et al. "A comprehensive approach to new physics simulations." The European Physical Journal C 71.2 (2011): 1-57.
[15] Jeary, A. P. "Damping in structures." Journal of wind engineering and industrial aerodynamics 72 (1997): 345- 355.