A Two-Dimensional Non-Relativistic Quantum Physics Game Engine

Luan P. Vargas[2], Marcelo R. Thielo[1], Luiz S. Martins-Filho[2], Ueslei Brandt[1]

[1]Universidade Federal do Pampa (Unipampa)
Av. Tiaraju, 810 , 97546-550 – Alegrete – RS – Brazil

[2]Universidade Federal do ABC (UFABC)
Al. da Universidade, s/n - Anchieta, 09606-045 – S. Bernardo do Campo, SP – Brazil

{marcelothielo,ueslei}@unipampa.edu.br, {luiz.martins,luan}@ufabc.edu.br

Abstract. *This paper presents an engine for creating interactive games using concepts from quantum mechanics, aiming to provide users with a tool capable of assisting in the creation of playable experiences that may help consolidate basic concepts in the field. We implemented a quantum simulation library based on the Schrödinger equation using the relaxation method. The prototype has a range of functions that already allow the creation of an interactive interface, in addition to quantum simulation capabilities. As a result, a prototype of a simple game was developed to demonstrate the current functionalities of the engine.*

Keywords— Quantum mechanics, Physics game engine, Schrödinger equation

1. Introduction

Physics engines play a crucial role in creating user-friendly and immersive virtual worlds in actual video games. Such engines allow the simulation of the laws of physics, enabling realistic interactions between objects, accurate collision detection, and natural movement. Over the years, the development of physics engines has significantly impacted the gaming industry, enhancing gameplay experiences and pushing the boundaries of realism, leaving sometimes a sensation of immersion on a completely different world.

The origins of physics engines in gaming can be traced back to the late 1980s and early 1990s [Dickman 2012]. During this period, games primarily relied on simple collision detection algorithms to handle basic interactions between objects. As game worlds became more and more complex, developers recognized the need for more advanced physics simulations. However, as the games' universes are usually macroscopic, the equations of physics implemented in those engines are only the classical ones. Considering the quantum theory, the authors in [Seskir 2022] present an extensive review of works on the application of quantum physics in games and interactive tools can be found. The authors in [Chiofalo 2022], and [Hoehn 2014] propose games based on quantum mechanics as tools for high schools and college levels physics teaching. Our work aims to contribute to fostering similar initiatives that utilize quantum physics in game development. This contribution involves innovating gameplay by exploring the potential of using occasionally counter-intuitive physics. Additionally, we aim to encourage contact with quantum theory, making it less intimidating and more stimulating, thus attracting the interest of students and the public in general.

In this work we developed, implemented and tested an experimental Quantum Physics Engine to simplify the development of games using quantum laws of interaction between elements.

Due to the high computational cost associated with three-dimensional quantum simulations, with approximately $O(n^3)$ per each three-dimensional "frame", where n is the number of elements on the grid edge, we created a two-dimensional version ($O(n^2)$ per frame) so to validate concepts and investigate other aspects like performance, potential interaction modes and entertainment factor. We performed simulations and generated visual results to provide insights on how such laws of physics could be explored interactively considering the aspect of entertaining. In the conclusions, we wrap up and propose some approaches to create entertaining games exploring quantum physics aspects by using our engine.

2. Solving the Schrödinger Equation

Since one of the main objectives of the present work is to provide a base tool for the development of applications capable of bringing a different vision to the use of physics in games, more specifically the use of quantum mechanics, the "core" of our engine is an algorithm that performs the numerical simulation of the time evolution of a given wave function according to the Schrödinger equation in two dimensions, it must be capable of generating dynamical output to a color interface.

During the development of the algorithm, convolution techniques were used so to obtain numerical solutions of the time-dependent Schrödinger equation [Eisberg and Resnick 1979], presented in the form of a partial differential equation given by Eq. 1.

$$ih\frac{@ \ (x;t)}{@t} = \frac{-h^2}{2m}\frac{@^2 \ (x;t)}{@x^2} + V(x) \tag{1}$$

where i = imaginary unit, h = Planck's reduced constant, m = mass of the particle, = complex wave function and V = time-independent potential.

This equation served as the basis for the quantum particles simulation proposed here. The wave function, which describes the state of the system, can be seen at Eq. 2.

$$(x;t=0) = (a\sqrt{\ })^{-\frac{1}{2}} e^{-(\frac{(x-x_0)}{2a})^2} e^{ixk_0} \tag{2}$$

where a = wavelength, x_0 = initial position of the particle and k_0 = momentum.

Therefore, given the wave function that describes the initial state of the particles, the user of our engine must iterate the Schrödinger equation by calling the method iterate() repeatedly. The engine will update each site on the grid, thus evolving the state of the system over time[Liu Guan-fu 2021]. The square of the complex conjugate of , which provides the probability density of the particle, $| \ |^2$, along with the fixed potential barriers $V(x)$ contains the information that will be displayed to the user at each frame. The user defines the position of the desired particle(s) and chooses the direction of movement of the wave functions. After that, the user can just push the green "Play" button to start the simulation.

3. Results and Discussion

We implemented a prototype of the engine as a C++ libray containing a class *QuantumGame*, running on Linux OS. That class could have been created using a singleton pattern due to computational cost of the simulation, but we did not want to impose any non-obvious restrictions to the developer so, if desired by the him or her, multiple different simulations can be instantiated and executed at the same time. The open source library SDL (Simple DirectMedia Layer) [Cysneiros and de Andrade 1993] was used for the graphical interface. Calculations and structures were implemented from scratch using standard C++ libraries.

The construction of the engine was structured with the intention of maximizing the freedom to the developer, so that all simulation parameters can be changed or redefined during the construction of the game. Furthermore, the convolutional implementation for solving the time-dependent Schrödinger equation (Eq. 1) allows the user to define arbitrary shaped, asymmetric barriers and in any screen position. In Fig. 1 a sketch of the system's classes is shown where *ComplexArray* and *Complex* are planned to be created after a planned refactoring of the system for better encapsulation. In Table 1, it is possible to see a list of methods for the main class responsible for the simulation.

Figure 1. Diagram for current classes (two at left) and to be included after refactoring (two at right)

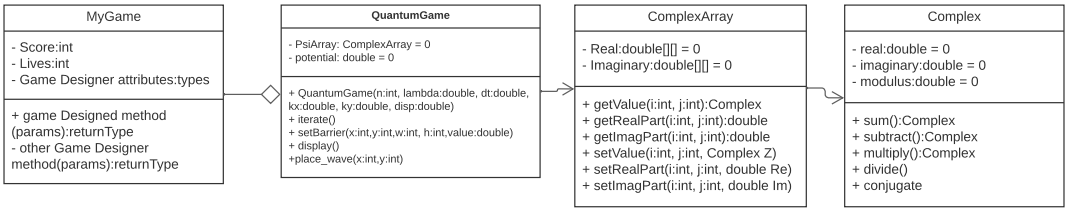


Table 1. Methods available in the QuantumGame class

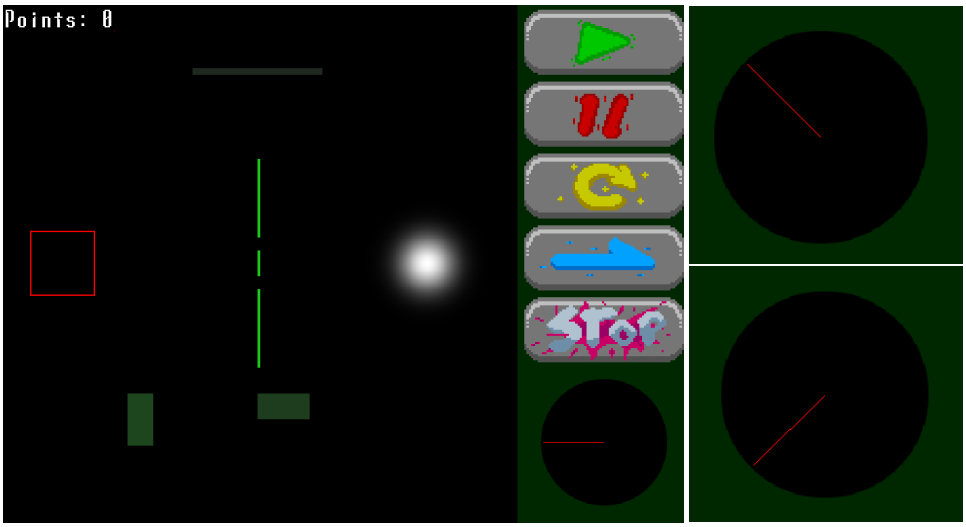
Method	Parameters	Description
QuantumGame()	**int n**: Defines the area size (NxN) of the simulation grid **double lambda**: Defines the initial particle wavelength for the simulation. **double dt**: Simulation time interval affects both simulating speed and accuracy. **double kx**: Particle momentum in X axis. **double ky**: Particle momentum in Y axis. **double disp**: Dispersion coefficient, sets the rate at which the energy of the propagation will disperse by each unity of distance.	Object Constructor, base for the game creation. Defines the initial simulation parameters. The values defined for k_x and k_y define the direction for the particle movement
void iterate()	This method has no parameters	Every time this method is called all the calculations are updated and time is increased by the amount of dt .
void set_barrier()	**int x**: Barrier position in X axis, relative to the simulation grid. **int y**: Barrier position in Y axis, relative to the simulation grid. **int w**: Barrier width. **int h**: Barrier height. **double value**: Potential energy of the barrier.	Defines the position, size and energy of the barrier that will be created on the grid.
void display()	This method has no parameters	This method launches the simulation display and the SDL library. So far, the user interface is being implemented within this method. However, in future versions, new methods will be implemented transparently to the developer.
void place_wave()	**int x**:Position on the X axis . **int y**: Position on the Y axis.	Resets the position of the particle after initialization.

The development was centered around a tool designed to enable developers to model interactive, real-time applications capable of simulating the behavior of wave packets[1] interacting with each other and with potential barriers. Our main focus is to bring a transparent development environment, without the need for a deep theoretical knowledge of quantum mechanics to build complex systems in a visual and interactive way. After finishing the construction of the *engine*, the sketch of a game was developed in order to demonstrate some possibilities and functions available in the interface. The game has a very simple gameplay that aims to show the user, in an intuitive way, the behavior of a quantum particle. It was developed with a golf-like game in mind, where a goal (red square) must be achieved to obtain a score. At the first moment (Fig. 2 left column), a screen with the initial state is shown to the player. In the menu on the right side of the screen, top to bottom we can see buttons with the following functionalities: Start (green), Pause (red), Reset (yellow), Apply Angle to Particle (blue) and, at the bottom, the Angle widget(circle with

[1] A wave packet is a concentrated train of (quantum) waves of various wavelengths or momenta confined within a small region of space.

red pointer) that allows the user to choose the initial angle for the particle movement (Fig. 2 right column). After choosing his/hers desired angle, the user must press the blue button so to apply the direction to the particle on screen. Currently, we only have the possibility of one single particle on the playfield but adding others in a near future should be straightforward.

Figure 2. Right: Test setup with a double slit barrier and other obstacles. Velocity vector of the single wave packet points from right to left parallel to x axis. Left: Two enlarged views of the angle selector depicting $+45^o$ and -45^o relatively to the x axis.



After choosing the desired angle, the simulation can be started by pressing the start button, so the player will visualize the animation of the particle displacing and interacting the predefined barriers (Fig. 3). In the sample game, the goal is to make the greatest amount of energy reach the red square at the end of the phase, then the player pushes the "Stop" button and sees his/her score.

A multiplayer version of the game is currently under development, introducing a strategic dimension to the gameplay. One player is tasked with defending a predefined perimeter using a limited amount of potential barriers. The objective is to thwart the second player, who will try to infiltrate the defended area by choosing the position and angle of launched particle. This dynamic compels both players to rely on their ability to build an intuition in predicting the quantum particle's trajectory and behavior. The defending player positions barriers strategically to obstruct the attacking player's progress. Simultaneously, the attacking player must evaluate the barriers and adjust the particle's launch angle to strategically optimize its path towards the goal.

4. Conclusion

According to the obtained results, we believe that our engine may provide a viable tool for the construction of games based on Quantum Mechanics. However, it still has performance limitations that can be overcome with parallel programming techniques, using libraries such as OpenMP or MPI, among others for CPU, or even Cuda or OpenACC if there is a need for GPU parallelization. Moreover, the parts concerning the functions for creating aesthetic layouts also need to be improved. Future versions should present a revision of the previously commented aspects. Future development should incorporate new game mechanic that highlights the classical path of the particle based on the probability density. This should implicitly inform the players about some of the principles of quantum mechanics through gameplay. The probability density function, which provides the likelihood of finding the particle in a particular location, could also be represented