



جامعة دمشق
كلية الهندسة المعلوماتية
السنة الثالثة

تقرير مشروع الحسابات العلمية



تقديم الطلاب :

- 1- محمد أحمد غزال.
- 2- أحمد حسين غزال فتح الله .
- 3- لجان أحمد عبدو

الفهرس :

..... المقدمة

..... فكرة المشروع

..... الهدف من المشروع

الفصل الاول

..... الدراسة الفيزيائية لحركة الكرة على سطح طاولة البلياردو.....

i. القوى الخارجيّة المؤثرة على الكرة

. قوة الثقل

. قوة رد الفعل

. القوى المعيقة لحركة الكرة

ii. الدراسة الحركية للكرة

. الحركة الانسحابية

. الحركة الدورانية

iii. اصطدام الكرة

الفصل الثاني

i. التوصيف البرمجي.....

الفصل الثالث

i. بنية المشروع.....

الفصل الرابع

i. التنفيذ البرمجي

الفصل الخامس

i. code

..... الخاتمة

..... المراجع

المقدمة:

إنّ التطور الحاصل في مجال العلوم و تكنولوجيا المعلومات وإمكانية الحصول على المعارف من مصادر مختلفة يضعنا أمام تحديات كبيرة لتطوير حياتنا اليومية وتقديم معرفة ملموسة تمكّننا من الاستفادة من علم الفيزياء في التطوير الحاصل في مجال التكنولوجيا، يهتم مشروعا بدراسة الحركة الفيزيائية الظاهرية لحركة كرة على سطح طاولة البلياردو حيث سنحاكي حركة هذه الكرة وسلوكها بشكل مفصّل ومفهوم.

فكرة المشروع:

- دراسة حركة الكرة الإنسحابية بدراسة القوى المؤثرة عليها.
- دراسة حركة الكرة الدورانية بدراسة القوى المؤثرة عليها (دوران الكرة حول نفسها).
- فلسفة هذه الدراسة من خلال: قوّة ضرب العصي - قوّة مقاومة الهواء - قوّة الاحتكاك .
- مناقشة عمليّة الصدم بالخشب وأيضاً صدم الكرات ببعضها البعض.

الهدف من المشروع:

دراسة الحركة الفيزيائية للكرة من وجهة نظر مراقب خارجي، عبر محاكاة حركة الكرة وسلوكها وتوازنها وكيفية التفاعل الحاصل بينها ومع الوسط المحيط مما يتيح للمبرمجين من تطوير هذه الفكرة وإضافة أفكار فيزيائية معقدة.

الفصل الاول

أولاً- القوى الخارجية المؤثرة على الكرة:

- تخضع الكرة الى مجموعة من القوى الخارجية المؤثرة عليها وهي:

1- قوّة الثقل :

وهي قوة شدتها تتعلق بتسارع الجاذبية الأرضية ويختلف هذا التسارع من منطقة إلى أخرى على سطح الأرض وقيمته عند خط الإستواء بمستوى سطح الأرض تساوي " $9.81m.s^{-2}$ " وبذلك فإن قوّة أي جسم تعطى بالعلاقة

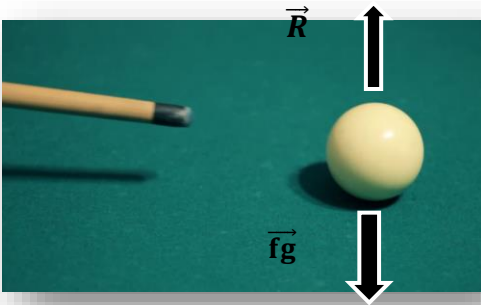
$$\vec{F}_g = m \cdot \vec{g}$$

حيث أن : m تمثل كتلة الكرة لدينا ووحدتها kg .

g ثابت الجاذبية الأرضية ووحدته $m.s^{-2}$.

F_g قوّة ثقل الكرة ووحدتها N .

-كتلة الكرة معلومة لدينا وتسارع الجاذبية ثابت بالنسبة لكل التجارب ومنهم نحصل على قوّة الثقل .



2- قوّة رد الفعل :

هي القوة المبذولة من الأرض على أي جسم متلامس معها وهنا قوّة رد الفعل ناتجة عن رد فعل الطاولة على الكرة .

ويرمز لها \vec{R} .

عناصرها :

-المبدأ : مركز ثقل الكرة .

-الحامل : الشاقول .

-الجهة : شعاع بعكس جهة الناظم على الكرة .

-الشدة :نفس قيمة قوّة الثقل .

- قوّة رد الفعل تساوي 0 لأن محورها عمودي على الحركة ويشكل معه زاوية 90° على سطح الطاولة .

3- القوى المعيقة لحركة الكرة :

- قوّة الإحتكاك :

ينشأ الإحتكاك من ملامسة أو إتصال جسم ما بسطح أو بجسم آخر.

في حالتنا يكون الاحتكاك ناتج عن ملامسة أو إتصال الكرة مع سطح الطاولة ويزداد الإحتكاك تبعاً لسطح الطاولة .

هناك نوعان من الإحتكاك:
السكوني و الحركي.

1- الإحتكاك السكوني:

هو القوة التي تمنع الجسم من التحرك عندما يكون ثابتاً.

• تعطى علاقة الإحتكاك السكوني بالشكل :

What are approximate values for pool equipment physical properties?

- ball diameter: 2.25 in
- ball mass: 6 oz
- ball mass moment of inertia: $2/5 mR^2$
- ball-ball coefficient of friction (μ): 0.03-0.08
- ball-ball coefficient of restitution (e): 0.92-0.98
- ball-cloth coefficient of rolling resistance (μ):
0.005 – 0.015
- ball-cloth coefficient of sliding friction (μ): 0.15-
0.4 (typical value: 0.2)
- ball-cloth spin deceleration rate: 5-15 rad/sec²
- ball-rail coefficient of restitution (e): 0.6-0.9
- ball-table coefficient of restitution (e): 0.5
- cue-tip-ball coefficient of friction (μ): 0.6
- cue-tip-ball coefficient of restitution (e): 0.71-
0.75 (leather tip), 0.81-0.87 (phenolic tip)

Here's a typical range of ball speeds used at the table.

Dr. Dave keeps this site **commercial free, with no ads**. If you appreciate the **free resources**, please consider making a one-time or monthly donation to **show your support**:

$$|F_{static}| \leq \mu_{static} * |r|$$

حيث :

μ_{static} : ثابت الإحتكاك بين السطحين والذي يتعلق بنوعية المادة المحتكة والذي يعبر عن النسبة بين قوة الإحتكاك بين الجسمين والقوة الضاغطة بينهما

F_{static} : قوة الإحتكاك المتولدة بين السطحين.

r : هي قوة رد الفعل باتجاه ناظم التلامس بين الجسم والسطح.

2- الإحتكاك الحركي:

إنّ الإحتكاك الحركي يتصرف بنفس طريقة الإحتكاك السكوني إنما مع إختلاف معامل الإحتكاك.

• تعطى علاقة الإحتكاك الحركي بالشكل:

$$|F_{dynamic}| \leq -\vec{v}_{planar} * \mu_{dynamic} * |r|$$

- جهة الاحتكاك هنا أصبحت بعكس جهة السرعة للجسم المتحرك.

- يعتبر الإحتكاك الحركي أقل تأثير من الإحتكاك السكوني.

- قوة مقاومة الهواء:

نوع من قوى الإحتكاك نلاحظها في الأشياء التي تتحرك بسرعات عالية وتؤثر عليها وتعوق حركتها ويكون تأثيرها كبيراً وواضحاً عندما تتحرك الاجسام بسرعات عالية وتقل عندما تتحرك الأجسام بسرعات منخفضة وعندما تتساوى مقدار قوة مقاومة الهواء مع القوى التي تتحرك فالقوى المؤثرة تكون متعادلة فيتحرك الجسم بسرعة ثابتة، وكلما ازدادت مساحة السطح المعرض للهواء ازداد مقدار مقاومة الهواء.

عناصرها:

- الحامل: نفس حامل شعاع السرعة.
- الجهة: بعكس جهة حركة الكرة.
- الشدة: $\frac{1}{2} * \text{معامل الاحتكاك} * \text{مساحة مسقط الكرة على مستوي عمودي على القوة} * \text{كثافة الهواء} * \text{مربع سرعة الجسم}.$

- العلاقة الشعاعية:

$$\vec{F}_d = - \frac{1}{2} \cdot C_d \cdot A \cdot \rho \cdot v^2$$

حيث:

- F_d : قوة مقاومة الهواء (قوة الإعاقة).
- ρ : كثافة الهواء وهو ثابت $\rho = 1.225$
- v : شعاع سرعة الجسم.
- C_d : معامل الإعاقة ليس له واحدة أو بعد.
- A : مساحة مسقط الكرة على مستوي عامودي على المركبة يتم حسابه.

حالات الكرة:

- حالة السكون: تنتج هذه الحالة عندما تكون الكرة ثابتة على سطح الطاولة دون حركة (أي انعدام القوى المؤثرة على الكرة).
- ويمكن التعبير عنها بالعلاقة الشعاعية كالتالي :

$$\sum \vec{F}_e = \vec{0}$$

$$\vec{F}_g + \vec{R} + \vec{F}_{static} + \vec{F}_d = \vec{0}$$

حيث:

- \vec{F}_g : قوّة ثابت الجاذبيّة الأرضيّة .
- \vec{R} : قوّة رد الفعل (وهي تتفانى مع قوّة الجاذبيّة حسب قانون نيوتن الثالث).
- \vec{F}_{static} : قوّة الاحتكاك السكوني.
- \vec{F}_d : قوّة مقاومة الهواء.

• حالة الحركة: تنتج هذه الحالة عندما تبدأ الكرة بالحركة على سطح الطاولة (أي عند إكساب الكرة طاقة حركية) ، ويمكن التعبير عن العلاقات المتعلقة بالحركة من خلال الدراسة الحركية للكرة.

ثانياً - الدراسة الحركية للكرة :

بدايةً عند ضرب العصا للكرة في مركزها يحدث تصادم مباشر لدينا يُكسب الكرة طاقة حركية تؤدي الى تحريك الكرة باتجاه مستقيم لأنّ الصدم مباشر فتتحرك الكرة بشكل انسحابي باتجاه شعاع العصا .

1- الحركة الإنسحابية :

نقول عن جسم أنه يتحرك حركة إنسحابية إذا كانت كافة نقاط الجسم تتحرك بنفس المسار وسرعة جميع نقاط الجسم متساوية في كل لحظة وبالتالي تُختار نقطة من نقاط هذا الجسم تُمثل حركته وهي مركز عطالته .

1.1 - مبدأ دالمبير:

- إن مبدأ دالمبير – المسمّى أحياناً مبدأ لاغرانج-دالمبير – هو من القوانين الأساسية في فيزياء الحركة الكلاسيكية.

- مبدأ دالمبير ينص على أنه في حال وجود عدّة قوى تؤثر على جسم بآنٍ واحد فإنه بإمكاننا استبدالها بقوة واحدة بحيث يكون:

$$\vec{F} = \sum_e \vec{F}_e$$

1.2 -العلاقة الأساسية في التحريك الإنسحابي (قانون نيوتن):

- إذا خضعت كرة كتلتها m الى قوى خارجية محصلتها $\sum \vec{F}_e$ فإن مركز عطالة الكرة يكتسب تسارعاً \vec{a}_c له في كل لحظة حامل وجهة $\sum \vec{F}_e$.

$$\sum \vec{F}_e = m \cdot \vec{a}_c$$

$$\vec{F}_g + \vec{R} + \vec{F}_{dynamic} + \vec{F}_d = m \cdot \vec{a}_c$$

*من هذه العلاقة يمكننا حساب التسارع كبداية لتحديث موضع نقاط الجسم بعد معرفة وتحصيل القوى المؤثرة عليه .

*إنَّ محصلة \vec{R} و \vec{F}_g أثناء السكون والحركة هي صفر لأنهما على حامل واحد وبجهتين متعاكستين (لكل فعل رد فعل يساويه بالشدة ويعاكسه بالاتجاه).

- تحديث معلومات الكرة أثناء الحركة الانسحابية :

لتحديث معلومات الكرة من (موقع ، سرعة ، تسارع) نحتاج الى تعريف مبدأ احداثيات الجسم الصلب .

أثناء الدراسة وعند التعامل مع الجسم فإن الجسم يغطّي عدة نقاط في آن واحد وبالتالي لا يمكننا أن نقول ان الجسم يغطّي النقطة (X,Y,Z) فلذلك نحن بحاجة الى نقطة تدلنا على هذا الجسم وهي مركز عطالة الجسم (الكرة) .

- التسارع :

من العلاقة الأساسية في التحريك وبالاستعانة بمبدأ دالمبير .

$$\vec{a} = \frac{1}{m} \vec{F}$$

- السرعة الخطية :

يمكننا تحديد السرعة الخطية في لحظة ما V' بدلالة سرعة في اللحظة السابقة V وتسارعه في اللحظة الحالية a من خلال العلاقة .

$$V' = v + a.t$$

- الموقع :

يتم تحديد موقع الكرة في لحظة ما بدلالة سرعتها وتسارعها في هذه اللحظة وموقعها في اللحظة السابقة من خلال العلاقة .

$$p' = p + v.t + \frac{1}{2}a.t^2$$

2- الحركة الدورانية :

نقول عن جسم أنه يتحرك حركة دورانية حول محور ثابت إذا كان لجميع نقاطه حركة دائرية حول ذلك المحور باستثناء النقاط التي تنتمي الى هذا المحور .

- نتجت لدينا الحركة الدورانية من احتكاك الكرة مع سطح الطاولة الذي يلامسها (دوران الكرة حول نفسها).

1.2 -العلاقة الأساسية في التحريك الدوراني :

- إذا دارت كرة عزم عطالتها I حول محور ثابت كان جداء تسارعه الزاوي α في عزم عطالته حول ذلك المحور مساويا العزم الحاصل للقوى الخارجية المؤثرة فيه، وبالتالي:

$$\sum \bar{J} = I \cdot \bar{\alpha}$$

J : عزم القصور الذاتي - I : عزم العطالة واحدها $(kg.m^2)$ - α : التسارع الزاوي واحده $(rad.s^{-2})$.

- بعد حساب التسارع الزاوي يمكن حساب التوجّه والسرعة الزاوية للأجسام وذلك من خلال تطبيق العلاقات التالية:

$$\omega' = \omega + \alpha.t$$

$$\theta' = \theta + \omega.t + \frac{1}{2} \alpha.t^2$$

حيث أن: ω السرعة الزاوية ، α التسارع الزاوي ، θ الموقع الزاوي.

- تحديث معلومات الكرة أثناء الحركة الدورانية :

للحصول على المعلومات التي تصف حركة الجسم الدورانية نقوم بتحديد التسارع الزاوي و السرعة الزاوية ثم الموقع.

- التسارع الزاوي :

* يتم حساب التسارع الزاوي من العلاقة الأساسية في التحريك الدوراني :

$$\bar{\alpha} = \frac{\sum \bar{J}}{I}$$

- السرعة الزاوية :

*يتم تحديث السرعة الزاوية للكرة في لحظة ما ω' بدلالة سرعته الزاوية في اللحظة السابقة ω وتسارعه الزاوي α في اللحظة الحالية من خلال العلاقة:

$$\omega' = \omega + \alpha.t$$

- الزاوية :

يتم تحديد الزاوية في لحظة ما Θ' بدلالة السرعة الزاوية في تلك اللحظة ω والزاوية في اللحظة السابقة Θ من خلال العلاقة:

$$\Theta' = \Theta + \omega.t$$

حساب سرعة وموقع وتسارع الكرة في كل لحظة:

-- لنفرض أن الكرة تحركت حركة إنسحابية ودورانية معاً، ونتجت عنها محصلة قوى $\sum \vec{F}_e$ ونريد أن نقوم بعملية تحديث للسرعة والموقع والتسارع .

يتم ذلك بحساب محصلة القوى وحفظها ضمن شعاع خاص يعبر عن محصلة القوى ، على أساس تلك المحصلة وبلاستفادة من العلاقة الأساسية في التحريك الإنسحابي سنحسب تسارع الكرة والذي بدوره يمكننا من حساب السرعة الجديدة للكرة لنحصل عن طريقها على موضع

$$p' = p + v.t + \frac{1}{2} a.t^2 \quad \text{الكرة الجديد وفق العلاقة:}$$

-- يعبر t عن الزمن الفاصل بين كل تحديثين متتاليين لانتقال الكرة وهو زمن صغير جداً وبما أن الحد المتعلق بالتسارع في العلاقة السابقة يتناسب طردياً مع نصف مربع الزمن $\frac{1}{2} a.t^2$ يمكن إهمال هذا الحد لتصبح العلاقة من الشكل:

$$p' = p + v.t$$

-- كذلك يمكن تحديث التسارع الزاوي والسرعة الزاوية والزاوية من خلال تطبيق القانون

$$\vec{v} = \vec{\omega} \cdot \vec{r} \quad \text{التالي:}$$

بذلك نحصل على السرعة الزاوية الموافقة للسرعة الخطية الجديدة ويتم حساب التسارع الزاوي من علاقة التسارع الزاوي والوصول أخيراً للزاوية من علاقة الزاوية أيضاً وذلك عند زمن معين .

ثالثاً - الصدم:

إن ترجمة الفيزياء للبلياردو تشمل في معظمها التصادم بين كراته .

عندما تصدم كرتا بلياردو ويكون التصادم مرناً تقريباً حيث أن التصادم المرن هو الذي تُحفظ خلاله الطاقة الحركية للنظام قبل وبعد التصادم .

في التصادم بين الكرات يبقى الإحتكاك موجوداً كما في أي تصادم آخر.

- لدينا تصادم بين العصا والكرة وهو الذي يسبب حركة الكرة بسرعة معينة واتجاه معين تبعاً للعصا والطاقة التي تمتلكها العصا (القوة المطبقة على الكرة من قبل العصا)

- تصادم بين كرة ثابتة وكرة متحركة حيث الكرة المتحركة تصطدم بالكرة الساكنة وتكسبها طاقة حركية .

- تصادم بين الكرة وحرف الطاولة حيث تصطدم الكرة بأحد حروف الطاولة الأربعة مما يؤدي الى تغير اتجاه الكرة بعكس زاوية الصدم .

- تصادم كرتين :

نفترض أن لدينا كرتين A,B نفس الكتلة وأن B في حالة سكون وA تتحرك .

نفترض أن السرعة الأولية ل A هي v_{1A} وبعد الصدم تتجه الكرة A بسرعة v_{2A} وتتحرك الكرة B بسرعة v_{2B} حيث $v_{1B}=0$ لان الكرة كانت في حالة سكون قبل الصدم .

حركة الكرة A في التصادم سيكون في اتجاه عمودي لاتجاه

الكرة B .

الطاقة الحركية لدينا محفوظة وسوف يتم التعبير عنها بالمعادلة

بشكل التالي :

$$* \quad 1/2m_A(v_{1A})^2 = 1/2m_A(v_{2A})^2 + 1/2m_B(v_{2B})^2$$

بعد التصادم وفي النقطة CP ، تتحرك الكرة B في اتجاه الخط الرابط بين مركزي الكرتين وهذا يعود الى القوة المستمدة من الكرة A نحو الكرة B وهكذا تتحرك B في اتجاه هذا الدافع .

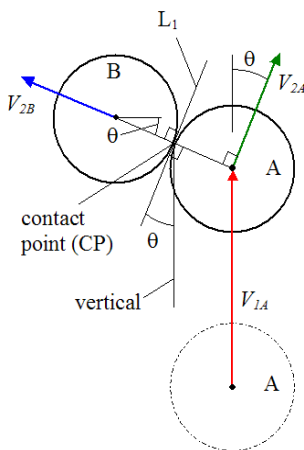
بالنسبة لمعادلة المتجهات العامة لحفظ الزخم الخطي تكون بالشكل :

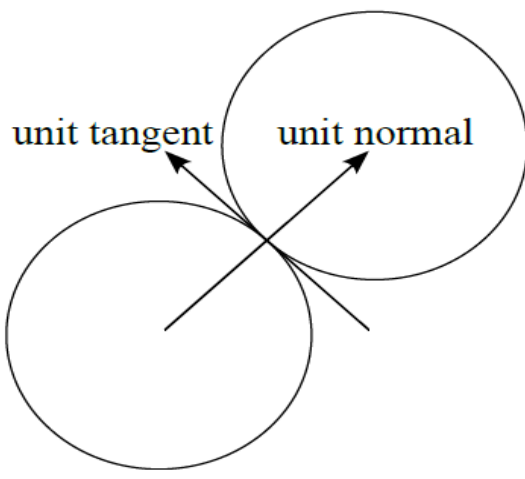
$$m_A \vec{v}_{1A} = m_A \vec{v}_{2A} + m_B \vec{v}_{2B}$$

وباختصار المعادلة (m_B و m_A متساويتين).

$$\vec{v}_{1A} = \vec{v}_{2A} + \vec{v}_{2B}$$

$$(v_{1A}) = (v_{2A}) + (v_{2B})$$





وحسب نظرية فيثاغورث ومن المعادلة الأخيرة فإن المتجهات V_{1A} و V_{2A} و V_{2B} تكون مثلثاً قائماً ومنه يمكننا تمثيل معادلة المتجهات لحفظ الزخم على الشكل الآتي.

- تصادم كرتين متحركتين (تصادمات مرنة في بعد واحد x, y):

في التصادم المرن يتم حفظ الطاقة كحرارة أو صوت أثناء الإصطدام. لا توجد تصادمات مثالية على نطاق واسع ، عند التصادم يتم حفظ كل الطاقة الحركية والزخم قبل وبعد الصدم .

الزخم هو نتاج الكتلة والسرعة : $p = mv$

$$m_1 \vec{v}_1 + m_2 \vec{v}_2 = m_1 \vec{v}_1' + m_2 \vec{v}_2'$$

الطاقة الحركية للكرة : $E_k = \frac{1}{2}mv^2$

$$1/2m_1v_1^2 + 1/2m_2v_2^2 = 1/2m_1v_1'^2 + 1/2m_2v_2'^2$$

الجمع بين معادلتى الزخم و الطاقة جبريا يعطي سرعات الكرات المتصادمة .

$$v_1' = \frac{v_1(m_1 - m_2) + 2m_2v_2}{m_1 + m_2} \quad v_2' = \frac{v_2(m_2 - m_1) + 2m_1v_1}{m_1 + m_2}$$

نوجد شعاع الناظم على الكرة : \vec{n}

$$\vec{n} = \langle x_2 - x_1, y_2 - y_1 \rangle$$

$$\vec{un} = \frac{\vec{n}}{|\vec{n}|} = \frac{\vec{n}}{\sqrt{n_x^2 + n_y^2}}$$

نوجد شعاع الواحدة الذي يمثل شعاع الناظم \vec{un}

$$\vec{ut} = \langle -un_y, un_x \rangle$$

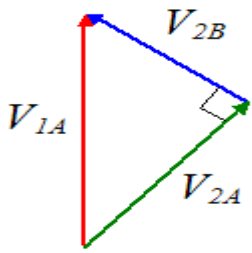
نوجد شعاع المماس \vec{u}_t

نسقط اشعة السرعة على اشعة المماس والناظم والتي يتم من خلالها اخذ سرعة الكرة بعد الصدم .

$$v_{2n} = \vec{un} \cdot \vec{v}_2 \quad v_{2t} = \vec{ut} \cdot \vec{v}_2$$

$$v_{1n} = \vec{un} \cdot \vec{v}_1 \quad v_{1t} = \vec{ut} \cdot \vec{v}_1 \quad \text{-العثور على السرعات التركيبية الجديدة بعد الصدم .}$$

-ايجاد السرعات العادية الجديدة .



$$v'_{1n} = \frac{v_{1n}(m_1 - m_2) + 2m_2 v_{2n}}{m_1 + m_2} \quad v'_{2n} = \frac{v_{2n}(m_2 - m_1) + 2m_1 v_{1n}}{m_1 + m_2}$$

نستخدم هذه الصيغ في الاصطدام أحادية الأبعاد .

-تحويل السرعات العنصرية العادية

$$\vec{v}'_1 = \vec{v}'_{1n} + \vec{v}'_{1t} \quad \vec{v}'_2 = \vec{v}'_{2n} + \vec{v}'_{2t}$$

والخالية من المتجهات الى سرعات بمتجهات .

$$v'_{1t} = v_{1t} \quad v'_{2t} = v_{2t}$$

$$\vec{v}'_{1n} = v'_{1n} \cdot \vec{un} \quad \vec{v}'_{1t} = v'_{1t} \cdot \vec{ut}$$

ايجاد السرعة النهائية : عن طريق جمع السرعة لناظم ومماس كل كرة .

$$\vec{v}'_{2n} = v'_{2n} \cdot \vec{un} \quad \vec{v}'_{2t} = v'_{2t} \cdot \vec{ut}$$

* عند انصدام كرتين بلياردو مع بعضهما نلاحظ ابتعادهما عن بعضهما في اتجاهين مختلفين بحيث تصنعان زاويتين مع اتجاه خط الحركة الابتدائي ويدعى ذلك بالتصادم ذي البعدين.

الفصل الثاني

البيئة البرمجية:

- بيئة التطوير المستخدمة والبرامج المستخدمة.
- لغة البرمجة C++ ضمن بيئة تطوير visual studio
- مكاتب ال Open GL

الفصل الثالث

بنية المشروع

بيئة الكرة :

وهي البنية الأساسية في مشروعنا
حيث لدينا Class Vector و Class Ball .

: Vector

وهو الصف الذي يمثل الكرات شعاعيا لدينا (x,y,z)
ويحوي بعض opearotor المساعدة في عمليات الجمع والطرح والضرب
والقسمة بين الكرات الممثلة شعاعيا .

: Ball

الصف يتم استدعاء Vector فيه وانشاء v,r والقوة والكتلة ورسم الكرة وتابع
التصادم واكتشاف التصادم والحركة الانسحابية.
يحوي التوابع :

FM() : يمثل قوة الثقل .

FR() : يمثل قوة رد الفعل.

Calc() : لتمثيل الحركة الانسحابية برمجيا .

Drow() : لرسم الكرة .

colDet() : تابع لاكتشاف الصدم بين كرتين .

Col() : تابع الصدم يحوي المعالجة الفيزيائية للصدم برمجيا.

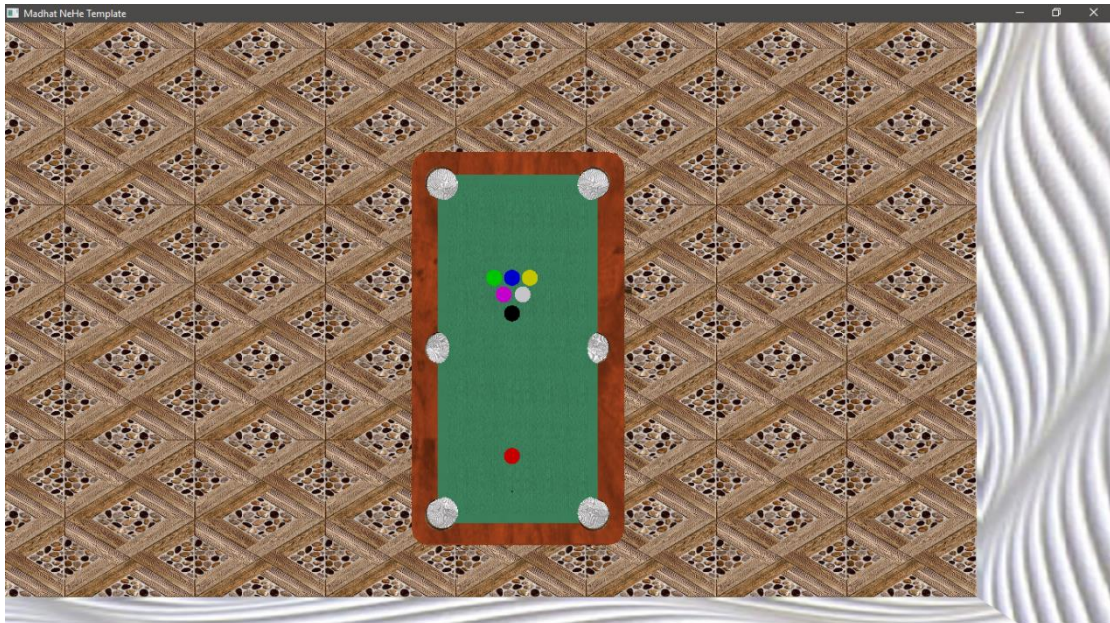
colWoo() : تابع صدم الكرة مع حرف الطاولة .

الفصل الرابع

التنفيذ

الحالة الأولى :

يكون فيه الكرات جاهزة static .



الحالة الثانية :

يتم ادخال سرعة وموضع وعدد الكرات الكرات dynamic .

الفصل الخامس

Code

كلاس ball():

```
#pragma once
#include "Vector.h"
class Ball
{
public:
    Vector r, v, a;
    double dt = 1.0 / 60.0;
    double m, red, green, blue, radius;
    Ball(){
        m = 1.0;
        radius = 0.68;

        red = 0;
        green = 0;
        blue = 0;
    }
    Vector FW(){
        Vector f;
        f.y = -9.8*m;
        return f;
    }
    Vector FR(){
        Vector f;
        f.y = +9.8*m;
        return f;
    }
    void calc(){
        //col();
        colwoo();
        Vector F = FW() + FR();
        a = F / m;
        v = a*dt + v;
        r = v*dt + r;
    }
    void draw(double x){
        radius = x;
        glPushMatrix();
        glTranslated(r.x, r.y, r.z);
        //drawQuad();
        glColor3f(red, green, blue);
        auxSolidSphere(radius);
    }
};
```

```

        glPopMatrix();
        glColor3f(1, 1, 1);
    }

    bool colDet(Ball b2){
        double dx = r.x - b2.r.x;
        double dy = r.y - b2.r.y;
        double dz = r.z - b2.r.z;
        double d2 = dx*dx + dy*dy + dz*dz;

        double d = sqrt(d2);

        if (d <= 2 * radius)
            return true;
        else
            return false;
    }

    void col(Ball& b){
        if (colDet(b) == true){
            Vector n, ut, un, v_11n, v_11t, v_22n, v_22t, v11, v22;
            double v1n, v1t, v2n, v2t, v_1n, v_2n, v_1t, v_2t;

            n = b.r - r;
            b.r = n*1.1 + r;

            un = n.norm();

            ut.x = -un.y;
            ut.y = un.x;

            v1n = un^v;
            v1t = ut^v;

            v2n = un^b.v;
            v2t = ut^b.v;

            v_1t = v1t;
            v_2t = v2t;

            v_1n = v1n*(m - b.m) + 2 * b.m * v2n / (m + b.m);
            v_2n = v2n*(b.m - m) + 2 * m * v1n / (m + b.m);

            v_11n = un * v_1n;
            v_11t = ut * v_1t;

            v_22n = un * v_2n;
            v_22t = ut * v_2t;

            v11 = v_11n + v_11t;
            v22 = v_22n + v_22t;

            v = v11;
            b.v = v22;
        }
    }

    void colwoo(){
        if (r.y < 0.3 && v.y < 0)
        {
            v.y = -v.y;
        }
        if (r.y > 27.7 && v.y > 0)

```

```

    {
        v.y = -v.y;
    }
    if (r.x < 0.6 && v.x < 0)
    {
        v.x = -v.x;
    }
    if (r.x > 12.4 && v.x > 0)
    {
        v.x = -v.x;
    }
    if (r.x <= 1 && r.y <= 1 || r.x >= 12 && r.y <= 1 || r.x <= 1 &&
r.y >= 27 || r.x >= 12 && r.y >= 27)
    {
        r.z = -114.5;
        v = 0;
    }
    if ((r.x <= 1 && (r.y >= 13 && r.y <= 15)) || (r.x >= 12 && (r.y
>= 13 && r.y <= 15)))
    {
        r.z = -114.5;
        v = 0;
    }
}

};

```

كلاس vector() :

```

#pragma once

#include <math.h>
#include <iostream>
using namespace std;

class Vector
{
public:
    double x, y, z;
    Vector(double xx = 0, double yy = 0, double zz = 0);
    void sett(double xx = 0, double yy = 0, double zz = 0);

    Vector operator+(const Vector& b);
    Vector operator-(const Vector& b);

    Vector operator*(const Vector& b);
    double operator^(const Vector& b);
    Vector operator*(const double& b);
    Vector operator/ (const double& b);

    Vector operator- ();

    double abs();
    Vector norm();

    void print();

    friend ostream& operator<< (ostream& os, const Vector& b);
    friend istream& operator>> (istream& is, Vector& b);
};

```

: Source

```
/*
 *      This Code Was Created By Jeff Molofee 2000
 *      A HUGE Thanks To Fredric Echols For Cleaning Up
 *      And Optimizing This Code, Making It More Flexible!
 *      If You've Found This Code Useful, Please Let Me Know.
 *      Visit My Site At nehe.gamedev.net
 */

#include <windows.h>           // Header File For Windows
#include <gl\gl.h>             // Header File For The OpenGL32 Library
#include <gl\glu.h>
#include <GLAUX.h>
#include <cmath>
#include <iostream>
#include <camera.h>
#include <GLTexture.h>
#include <Model_3DS.h>
#include <texture.h>
#include "Ball.h"

int nBalls = 8;
Ball balls[50];
using namespace std;

HDC          hDC = NULL;      // Private GDI Device Context
HGLRC        hRC = NULL;      // Permanent Rendering Context
HWND         hWnd = NULL;      // Holds Our Window Handle
HINSTANCE     hInstance;      // Holds The Instance Of The Application

bool keys[256];               // Array Used For The Keyboard Routine
bool active = TRUE;           // Window Active Flag Set To TRUE By Default
bool fullscreen = TRUE;      // Fullscreen Flag Set To Fullscreen Mode By Default

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM); // Declaration For WndProc
Camera MyCamera;
GLvoid ReSizeGLScene(GLsizei width, GLsizei height) // Resize And Initialize The GL Window
{
    if (height == 0)
    {
        // Prevent A Divide By Zero By
        height = 1;
        // Making Height Equal One
    }

    glViewport(0, 0, width, height); // Reset The Current Viewport

    glMatrixMode(GL_PROJECTION); // Select The Projection Matrix
    glLoadIdentity();
    // Reset The Projection Matrix

    // Calculate The Aspect Ratio Of The Window
    gluPerspective(45.0f, (GLfloat)width / (GLfloat)height, 0.1f, 1000.0f);
}
```

```

        glMatrixMode(GL_MODELVIEW); //
Select The Modelview Matrix
        glLoadIdentity();// Reset The Modelview Matrix
    }
    int fac1;
    GLUquadric * quadric;
    GLUquadric * m;

    Model_3DS M; //-----
    GLTexture m1, m2, m3, m4, m5, m6;
    int img, img1;

    void physics(){
        for (int i = 0; i < nBalls - 1; i++){
            {
                balls[i].calc();
                balls[i].draw(0.68);
            }
            balls[7].calc();
            balls[7].draw(0.068);
        }

        /*
        for (int i = 0; i < 3; i++){
            for (int j = 2; j > 0; j--){
                if (balls[i].colDet(balls[j])){
                    balls[i].v = -balls[i].v;
                    balls[j].v = -balls[j].v;
                }
            }
        }
        */

        for (int i = 0; i < nBalls; i++){
            for (int j = i + 1; j < nBalls; j++){
                {
                    balls[i].col(balls[j]);
                }
            }
        }

    void initBalls(){
        //red
        balls[0].r.sett(6, 5, -111.5); //down
        //balls[0].v.sett(0, 30);

        //green
        balls[1].r.sett(4.5, 20, -111.5); //right
        //balls[1].v.sett(30, 10);

        //blue
        balls[2].r.sett(6, 20, -111.5); //up
        //balls[2].v.sett(0, -30);

        //yellow
        balls[3].r.sett(7.5, 20, -111.5); //left
        //balls[3].v.sett(30, 20);

        //roz
        balls[4].r.sett(5.3, 18.6, -111.5);
        //balls[4].v.sett(0, 30, 0);
    }

```

```

//wite
balls[5].r.sett(6.9, 18.6, -111.5);
//balls[6].v.sett(0, 30, 0);

```

```

//balck
balls[6].r.sett(6, 17, -111.5);
//balls[7].v.sett(0, 30, 0);

```

```

//stick
balls[7].r.sett(6, 2, -111.5);
//balls[5].v.sett(0, 50, 0);

```

```

balls[0].red = 1;
balls[1].green = 1;
balls[2].blue = 1;
balls[3].red = 1;
balls[3].green = 1;
balls[4].red = 1;
balls[4].blue = 1;
balls[5].red = 1;
balls[5].green = 1;
balls[5].blue = 1;
/*balls[6].red = 1;
balls[6].green = 1;
balls[6].blue = 1;*/

```

```

}
double n, x[100], y[100], d[100], f[100], v[100];
void set_ball(){

```

```

    cout << "please enter the num of balls" << endl;
    cin >> n;
    for (int i = 0; i < n; i++){
        cout << "pleas enter the position ball" << i << " " << "ex:
(X,Y)" << "the domain x[0->13],the domain y[0->28] " << endl;
        cin >> x[i] >> y[i];
    }
    cout << "please enter the speed of balls" << endl;
    for (int i = 0; i < n; i++){
        cout << "pleas enter the speed ball" << i << " " << "ex: (X,Y)"
<< endl;
        cin >> d[i] >> f[i];
    }
    cout << "please enter the valum of balls" << endl;
    for (int i = 0; i < n; i++){
        cout << "pleas enter the valum ball" << i << " " << "ex: (V)" <<
endl;
        cin >> v[i];
    }
    for (int i = 0; i < n; i++)
    {
        balls[i].r.x = x[i];
        balls[i].r.y = y[i];
        balls[i].r.z = -111.5;

        balls[i].v.x = d[i];
        balls[i].v.y = f[i];

        balls[i].red = i;
        balls[i].green = i - 2;
        balls[i].blue = i - 3;
    }
}

```

```

    }
}

void print_ball(){
    for (int i = 0; i < n; i++)
    {
        balls[i].calc();
        balls[i].draw(v[i]);
    }
    for (int i = 0; i < n; i++){
        for (int j = i + 1; j < n; j++)
        {
            balls[i].col(balls[j]);
        }
    }
}

bool a;
int InitGL(GLvoid)
// All Setup For OpenGL Goes Here
{
    cin >> a;
    if (a)
    {
        set_ball();
    }
    else
    {
        initBalls();
    }
}

MyCamera = Camera();
glShadeModel(GL_SMOOTH); //
Enable Smooth Shading
glClearColor(0.0f, 0.0f, 0.0f, 0.5f); // Black
Background
glClearDepth(1.0f);
// Depth Buffer Setup
glEnable(GL_DEPTH_TEST); //
Enables Depth Testing
glDepthFunc(GL_LEQUAL);
// The Type Of Depth Testing To Do
glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST); // Really Nice
Perspective Calculations

M = Model_3DS();
M.Load("Pool Table.3DS");
glPushMatrix();
glEnable(GL_TEXTURE_2D);
m1.LoadBMP("r1.bmp");
m2.LoadBMP("r2.bmp");
m3.LoadBMP("r3.bmp");
m4.LoadBMP("r4.bmp");
m5.LoadBMP("r5.bmp");
m6.LoadBMP("r6.bmp");
glPopMatrix();
//glTranslated(0, 0, -15);
//glRotated(t, 0, 1, 0);

```

```

//glTranslated(0, 0, 15);
glPushMatrix();
glEnable(GL_PROXY_TEXTURE_2D);
img = LoadTexture("ground.bmp", 255);
img1 = LoadTexture("woolffront.bmp", 255);
glPopMatrix();

```

```

M.Materials[0].tex = m2;//--
M.Materials[1].tex = m5;/--
M.Materials[2].tex = m1;/--
M.Materials[3].tex = m4;
M.Materials[4].tex = m6;
M.Materials[5].tex = m2;

```

```

M.scale = 0.3;
M.rot.z = 90;
M.rot.y = 90;

```

```

M.pos.x = 6.5;
M.pos.y = 9.5;
M.pos.z = -120;

```

```

//M.pos.x = -4;
//M.pos.y = -10;
//M.pos.z = -50;

```

```

//glClear(GL_COLOR_BUFFER_BIT);
return TRUE;
// Initialization Went OK

```

```

}

```

```

void ground1(){

```

```

    glPushMatrix();
    glPushMatrix();
    glBindTexture(GL_TEXTURE_2D, img1);
    glBegin(GL_QUADS);
    glTexCoord2d(0, 0);
    glVertex3d(-50, 0, 20);
    glTexCoord2d(1, 0);
    glVertex3d(-50, 0, -120);
    glTexCoord2d(1, 1);
    glVertex3d(50, 0, -120);
    glTexCoord2d(0, 1);
    glVertex3d(50, 0, 20);
    glPopMatrix();
    glEnd();

```

```

    glPushMatrix();
    glBindTexture(GL_TEXTURE_2D, img1);
    glBegin(GL_QUADS);
    glTexCoord2d(0, 0);
    glVertex3d(-50, 0, 20);
    glTexCoord2d(1, 0);
    glVertex3d(-50, 60, 20);
    glTexCoord2d(1, 1);
    glVertex3d(-50, 60, -120);
    glTexCoord2d(0, 1);
    glVertex3d(-50, 0, -120);
    glPopMatrix();
    glEnd();

```



```

    glPushMatrix();
    glBindTexture(GL_TEXTURE_2D, img);
    glBegin(GL_QUADS);
    glTexCoord2d(0, 0);
    glVertex3d(-50, 60, -120);
    glTexCoord2d(8, 0);
    glVertex3d(50, 60, -120);
    glTexCoord2d(8, 8);
    glVertex3d(50, 0, -120);
    glTexCoord2d(0, 8);
    glVertex3d(-50, 0, -120);
    glEnd();
    glPopMatrix();

```

```

    glPushMatrix();
    glBindTexture(GL_TEXTURE_2D, img1);
    glBegin(GL_QUADS);
    glTexCoord2d(0, 0);
    glVertex3d(50, 0, 20);
    glTexCoord2d(1, 0);
    glVertex3d(50, 0, -120);
    glTexCoord2d(1, 1);
    glVertex3d(50, 60, -120);
    glTexCoord2d(0, 1);
    glVertex3d(50, 60, 20);
    glEnd();
    glPopMatrix();

```

```

    glPushMatrix();
    glBindTexture(GL_TEXTURE_2D, img1);
    glBegin(GL_QUADS);
    glTexCoord2d(0, 0);
    glVertex3d(-50, 60, 20);
    glTexCoord2d(1, 0);
    glVertex3d(50, 60, 20);
    glTexCoord2d(1, 1);
    glVertex3d(50, 60, -120);
    glTexCoord2d(0, 1);
    glVertex3d(-50, 60, -120);
    glEnd();
    glPopMatrix();

```

```

    glFlush();
    glPopMatrix();

```

```

}

```

```

void drawQuad(){
    glBegin(GL_QUADS);
    glColor3f(1, 0, 0);
    glVertex3d(-1, -1, 0);
    glVertex3d(-1, 1, 0);

```

```

    glColor3f(0, 0, 1);
    glVertex3d(1, 1, 0);
    glVertex3d(1, -1, 0);
    glEnd();

```

```

    glColor3f(1, 1, 1);

```

```

}
float angle = 0;
double t;
bool drawTableModel = true;

void DrawGLScene(GLvoid)
    // Here's Where We Do All The Drawing
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);    // Clear Screen
    And Depth Buffer
    glLoadIdentity();    // Reset The Current Modelview Matrix
    MyCamera.Render();
    //glScalef(0.1, 0.1, 0.1);
    if (keys['S'])        MyCamera.MoveForward(-0.8);
    if (keys['W'])        MyCamera.MoveForward(0.8);
    if (keys[VK_RIGHT])    MyCamera.RotateY(-1.4);
    if (keys[VK_LEFT])    MyCamera.RotateY(1.4);
    if (keys[VK_DOWN])    MyCamera.MoveUpward(-0.2);
    if (keys[VK_UP])    MyCamera.MoveUpward(0.2);
    if (keys['A'])        MyCamera.MoveRight(-0.8);
    if (keys['D'])        MyCamera.MoveRight(0.8);

    /*if (keys['M'])
        M.scale += 0.05;
        if (keys['N'])
            M.scale -= 0.05;
    */
    M.Draw();
    glPushMatrix();
    glTranslated(0, -10, 0);
    ground1();
    glPopMatrix();

    if (a){
        print_ball();
    }
    else
    {
        physics();
    }

    if (keys['Z']){
        balls[7].v.sett(0, 75, 0);
    }
    if (keys['X']){
        balls[7].r.z = -115.5;
        balls[7].v = 0;
    }
}
//DO NOT REMOVE THIS
SwapBuffers(hDC);
glFlush();

}

GLvoid KillGLWindow(GLvoid)    //
    Properly Kill The Window
{
    if (fullscreen)
        // Are We In Fullscreen Mode?
        {
            ChangeDisplaySettings(NULL, 0);    //
            If So Switch Back To The Desktop
        }
}

```

```

        ShowCursor(TRUE);
        // Show Mouse Pointer
    }

    if (hRC)
        // Do We Have A Rendering Context?
    {
        if (!wglMakeCurrent(NULL, NULL))
            // Are We Able To Release The DC And RC Contexts?
        {
            MessageBox(NULL, "Release Of DC And RC Failed.", "SHUTDOWN
ERROR", MB_OK | MB_ICONINFORMATION);
        }

        if (!wglDeleteContext(hRC))
            // Are We Able To Delete The RC?
        {
            MessageBox(NULL, "Release Rendering Context Failed.",
"SHUTDOWN ERROR", MB_OK | MB_ICONINFORMATION);
        }
        hRC = NULL;
        // Set RC To NULL
    }

    if (hDC && !ReleaseDC(hWnd, hDC))
        // Are We
Able To Release The DC
    {
        MessageBox(NULL, "Release Device Context Failed.", "SHUTDOWN
ERROR", MB_OK | MB_ICONINFORMATION);
        hDC = NULL;
        // Set DC To NULL
    }

    if (hWnd && !DestroyWindow(hWnd))
        // Are We
Able To Destroy The Window?
    {
        MessageBox(NULL, "Could Not Release hWnd.", "SHUTDOWN ERROR",
MB_OK | MB_ICONINFORMATION);
        hWnd = NULL;
        // Set hWnd To NULL
    }

    if (!UnregisterClass("OpenGL", hInstance))
        // Are We
Able To Unregister Class
    {
        MessageBox(NULL, "Could Not Unregister Class.", "SHUTDOWN ERROR",
MB_OK | MB_ICONINFORMATION);
        hInstance = NULL;
        // Set hInstance To NULL
    }
}

/* This Code Creates Our OpenGL Window. Parameters Are:
*
* title - Title To Appear At The Top Of The Window
*
* width - Width Of The GL Window Or Fullscreen Mode
*
* height - Height Of The GL Window Or Fullscreen Mode
*
* bits - Number Of Bits To Use For Color (8/16/24/32)
*
*/

```

```

*   fullscreenflag   - Use Fullscreen Mode (TRUE) Or Windowed Mode
(FALSE)   */

BOOL CreateGLWindow(char* title, int width, int height, int bits, bool
fullscreenflag)
{
    GLuint PixelFormat;           // Holds The Results After
Searching For A Match
    WNDCLASS wc;                  // Windows Class
    Structure
        DWORD dwExStyle;         // Window Extended Style
        DWORD dwStyle;           // Window Style
        RECT WindowRect;          // Grabs Rectangle Upper
Left / Lower Right Values
        WindowRect.left = (long)0; // Set Left Value To 0
        WindowRect.right = (long)width; // Set Right Value To Requested
Width
        WindowRect.top = (long)0; // Set Top Value To 0
        WindowRect.bottom = (long)height; // Set Bottom Value To
Requested Height

    fullscreen = fullscreenflag; // Set The Global
Fullscreen Flag

    hInstance = GetModuleHandle(NULL); // Grab An
Instance For Our Window
    wc.style = CS_HREDRAW | CS_VREDRAW | CS_OWNDC; // Redraw On Size, And
Own DC For Window.
    wc.lpfnWndProc = (WNDPROC)WndProc; // WndProc
Handles Messages
    wc.cbClsExtra = 0;
    // No Extra Window Data
    wc.cbWndExtra = 0;
    // No Extra Window Data
    wc.hInstance = hInstance; //
Set The Instance
    wc.hIcon = LoadIcon(NULL, IDI_WINLOGO); // Load The
Default Icon
    wc.hCursor = LoadCursor(NULL, IDC_ARROW); // Load The Arrow
Pointer
    wc.hbrBackground = NULL;
    // No Background Required For GL
    wc.lpszMenuName = NULL;
    // We Don't Want A Menu
    wc.lpszClassName = "OpenGL";
    // Set The Class Name

    if (!RegisterClass(&wc))
    {
        // Attempt To Register The Window Class
        MessageBox(NULL, "Failed To Register The Window Class.", "ERROR",
MB_OK | MB_ICONEXCLAMATION);
        return FALSE;
        // Return FALSE
    }

    if (fullscreen)
    {
        // Attempt Fullscreen Mode?
        DEVMODE dmScreenSettings;
        // Device Mode

```

```

        memset(&dmScreenSettings, 0, sizeof(dmScreenSettings)); // Makes
Sure Memory's Cleared
        dmScreenSettings.dmSize = sizeof(dmScreenSettings); //
Size Of The Devmode Structure
        dmScreenSettings.dmPelsWidth = width; //
Selected Screen Width
        dmScreenSettings.dmPelsHeight = height; //
Selected Screen Height
        dmScreenSettings.dmBitsPerPel = bits;
        // Selected Bits Per Pixel
        dmScreenSettings.dmFields = DM_BITSPERPEL | DM_PELSWIDTH |
DM_PELSHHEIGHT;

        // Try To Set Selected Mode And Get Results. NOTE:
CDS_FULLSCREEN Gets Rid Of Start Bar.
        if (ChangeDisplaySettings(&dmScreenSettings, CDS_FULLSCREEN) !=
DISP_CHANGE_SUCCESSFUL)
        {
            // If The Mode Fails, Offer Two Options. Quit Or Use
Windowed Mode.
            if (MessageBox(NULL, "The Requested Fullscreen Mode Is Not
Supported By\nYour Video Card. Use Windowed Mode Instead?", "NeHe GL", MB_YESNO
| MB_ICONEXCLAMATION) == IDYES)
            {
                fullscreen = FALSE; // Windowed Mode
Selected. Fullscreen = FALSE
            }
            else
            {
                // Pop Up A Message Box Letting User Know The
Program Is Closing.
                MessageBox(NULL, "Program Will Now Close.",
"ERROR", MB_OK | MB_ICONSTOP);
                return FALSE;
            }
            // Return FALSE
        }
    }

    if (fullscreen)
    {
        // Are We Still In Fullscreen Mode?
        {
            dwExStyle = WS_EX_APPWINDOW;
            // Window Extended Style
            dwStyle = WS_POPUP;
            // Windows Style
            ShowCursor(FALSE);
            // Hide Mouse Pointer
        }
    }
    else
    {
        dwExStyle = WS_EX_APPWINDOW | WS_EX_WINDOWEDGE; //
Window Extended Style
        dwStyle = WS_OVERLAPPEDWINDOW;
        // Windows Style
    }

    AdjustWindowRectEx(&WindowRect, dwStyle, FALSE, dwExStyle); //
Adjust Window To True Requested Size

    // Create The Window

```

```

        if (!(hWnd = CreateWindowEx(dwExStyle,
            // Extended Style For The Window
            "OpenGL", // Class
Name
            title, // Window
Title
            dwStyle | // Defined
Window Style
            WS_CLIPSIBLINGS | // Required
Window Style
            WS_CLIPCHILDREN, // Required
Window Style
            0, 0, // Window
Position
            WindowRect.right - WindowRect.left, // Calculate Window
Width
            WindowRect.bottom - WindowRect.top, // Calculate Window
Height
            NULL, // No
Parent Window
            NULL, // No Menu
            hInstance, //
Instance
            NULL))) //
Dont Pass Anything To WM_CREATE
{
    KillGLWindow();
    // Reset The Display
    MessageBox(NULL, "Window Creation Error.", "ERROR", MB_OK |
MB_ICONEXCLAMATION);
    return FALSE; //
Return FALSE
}

```

```

static PIXELFORMATDESCRIPTOR pfd = // pfd Tells
Windows How We Want Things To Be
{
    sizeof(PIXELFORMATDESCRIPTOR), // Size Of
This Pixel Format Descriptor
    1, // Version Number
    PFD_DRAW_TO_WINDOW | // Format
Must Support Window
    PFD_SUPPORT_OPENGL | // Format
Must Support OpenGL
    PFD_DOUBLEBUFFER, //
Must Support Double Buffering
    PFD_TYPE_RGBA,
    // Request An RGBA Format
    bits,
    // Select Our Color Depth
    0, 0, 0, 0, 0, 0, //
Color Bits Ignored
    0,
    // No Alpha Buffer
    0,
    // Shift Bit Ignored
    0,
    // No Accumulation Buffer
    0, 0, 0, 0,
    // Accumulation Bits Ignored

```

```

        16,
        // 16Bit Z-Buffer (Depth Buffer)
        0,
        // No Stencil Buffer
        0,
        // No Auxiliary Buffer
        PFD_MAIN_PLANE,
    // Main Drawing Layer
    0,
    // Reserved
    0, 0, 0
    // Layer Masks Ignored
};

    if (!(hDC = GetDC(hWnd))) //
Did We Get A Device Context?
    {
        KillGLWindow();
        // Reset The Display
        MessageBox(NULL, "Can't Create A GL Device Context.", "ERROR",
MB_OK | MB_ICONEXCLAMATION);
        return FALSE; //
    Return FALSE
    }

    if (!(PixelFormat = ChoosePixelFormat(hDC, &pfd))) // Did Windows
Find A Matching Pixel Format?
    {
        KillGLWindow();
        // Reset The Display
        MessageBox(NULL, "Can't Find A Suitable PixelFormat.", "ERROR",
MB_OK | MB_ICONEXCLAMATION);
        return FALSE; //
    Return FALSE
    }

    if (!SetPixelFormat(hDC, PixelFormat, &pfd)) // Are We Able To
Set The Pixel Format?
    {
        KillGLWindow();
        // Reset The Display
        MessageBox(NULL, "Can't Set The PixelFormat.", "ERROR", MB_OK |
MB_ICONEXCLAMATION);
        return FALSE; //
    Return FALSE
    }

    if (!(hRC = wglCreateContext(hDC)) // Are We
Able To Get A Rendering Context?
    {
        KillGLWindow();
        // Reset The Display
        MessageBox(NULL, "Can't Create A GL Rendering Context.", "ERROR",
MB_OK | MB_ICONEXCLAMATION);
        return FALSE; //
    Return FALSE
    }

    if (!wglMakeCurrent(hDC, hRC)) // Try To
Activate The Rendering Context
    {

```

```

        KillGLWindow();
        // Reset The Display
        MessageBox(NULL, "Can't Activate The GL Rendering Context.",
"ERROR", MB_OK | MB_ICONEXCLAMATION);
        return FALSE;
Return FALSE
    }

    ShowWindow(hWnd, SW_SHOW);
The Window
    SetForegroundWindow(hWnd);
Slightly Higher Priority
    SetFocus(hWnd);
    // Sets Keyboard Focus To The Window
    ReSizeGLScene(width, height);
Our Perspective GL Screen

    if (!InitGL())
    // Initialize Our Newly Created GL Window
    {
        KillGLWindow();
        // Reset The Display
        MessageBox(NULL, "Initialization Failed.", "ERROR", MB_OK |
MB_ICONEXCLAMATION);
        return FALSE;
Return FALSE
    }

    return TRUE;
Success
}

LRESULT CALLBACK WndProc(HWND hWnd,
Window
    UINT uMsg,
WPARAM wParam,
LPARAM lParam)
{
    static PAINTSTRUCT ps;

    switch (uMsg)
    {
        case WM_PAINT:
            DrawGLScene();
            BeginPaint(hWnd, &ps);
            EndPaint(hWnd, &ps);
            return 0;

        case WM_TIMER:
            DrawGLScene();
            return 0;

        case WM_ACTIVATE:
            if (!HIWORD(wParam))
            {
                // Watch
                // Check
                // Minimization State
            }
    }
}

```



```

        active = TRUE; //
Program Is Active
    }
    else
    {
        active = FALSE; //
Program Is No Longer Active
    }

    return 0; //
Return To The Message Loop
}

    case WM_SYSCOMMAND: //
Intercept System Commands
    {
        switch (wParam) //
Check System Calls
        {
            case SC_SCREENSAVE: // Screensaver
Trying To Start?
            case SC_MONITORPOWER: // Monitor Trying
To Enter Powersave?
                return 0; //
Prevent From Happening
        }
        break; //
Exit
    }

    case WM_CLOSE: //
Did We Receive A Close Message?
    {
        PostQuitMessage(0); // Send A
Quit Message
        return 0; //
Jump Back
    }

    case WM_KEYDOWN: // Is A
Key Being Held Down?
    {
        keys[wParam] = TRUE; // If So, Mark It
As TRUE

```

```

        return 0; //
    }

    case WM_KEYUP: //
    Has A Key Been Released?
    {
        keys[wParam] = FALSE; // If So,
    Mark It As FALSE
    }

    return 0; //
    }

    case WM_SIZE: // Resize
    The OpenGL Window
    {
        ReSizeGLScene(LOWORD(lParam), HIWORD(lParam)); // LowWord=Width,
    HiWord=Height
    }

    return 0; //
    }

    // Pass All Unhandled Messages To DefWindowProc
    return DefWindowProc(hWnd, uMsg, wParam, lParam);
}

int mmmm(HINSTANCE hInstance, // Instance
        HINSTANCE hPrevInstance, // Previous Instance
        LPSTR lpCmdLine, // Command Line Parameters
        int nCmdShow) // Window Show State
{
    MSG msg;
    // Windows Message Structure
    BOOL done = FALSE; //
    Bool Variable To Exit Loop

    // Ask The User Which Screen Mode They Prefer
    //if (MessageBox(NULL, "Would You Like To Run In Fullscreen Mode?",
    "Start FullScreen?", MB_YESNO|MB_ICONQUESTION)==IDNO)
    {
        fullscreen = FALSE; //
    }

    // Create Our OpenGL Window
    if (!CreateGLWindow("Madhat NeHe Template", 640, 480, 16, fullscreen))
    {
        return 0;
    }
    // Quit If Window Was Not Created

    //Set drawing timer to 20 frame per second
    UINT timer = SetTimer(hWnd, 0, 50, (TIMERPROC)NULL);

    while (GetMessage(&msg, NULL, 0, 0))

```

```

    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return 0;
}

int main(HINSTANCE hInstance,          // Instance
        HINSTANCE hPrevInstance,      // Previous Instance
        LPSTR lpCmdLine,              // Command Line Parameters
        int nCmdShow)
{
    return mmmm(hInstance,             // Instance
                hPrevInstance,         // Previous Instance
                lpCmdLine,              // Command Line Parameters
                nCmdShow);
}

////////////////////////////////////

void chooseColorForAddBallDlg(HWND hWnd) {
    static COLORREF customColors[16]; // Array to hold custom colors used
    for "Choose Color" dialog
    CHOOSECOLOR cc;
    cc.lStructSize = sizeof(cc);
    cc.hwndOwner = hWnd;
    //cc.rgbResult = g_bAdd.color(); // Select color of last ball created
    cc.lpCustColors = customColors;
    cc.Flags = CC_RGBINIT; // Initialize color to value of rgbResult
    BOOL ccRetVal = ChooseColor(&cc);
    //if (ccRetVal) g_bAdd.setColor(cc.rgbResult);
    //if (GetRValue(cc.rgbResult) > LIGHT_COLOR_THRESHOLD &&
    GetGValue(cc.rgbResult) > LIGHT_COLOR_THRESHOLD && GetBValue(cc.rgbResult) >
    LIGHT_COLOR_THRESHOLD) {
        //MessageBox(hWnd, "Note: Light colored balls may be difficult to
        see.\r\nYou might wish to choose a different color.", "Note",
        MB_ICONINFORMATION);
    }
}




```

الخاتمة:

كما رأيت يمكن أن تتدخل فيزياء البلياردو بشكل قوي عند اعتبار كل ما قد يحدث في لعبة نموذجية.

بإمكاننا المراهنة على اعتماد اللاعبين المحترفين على الفيزياء الموجودة هنا خلال ممارسة هذه اللعبة.

المراجع المستخدمة:

www.real-world-physics-problems.com 
www.hyperphysics.phy-astr.gsu.edu.com 
www.physicsacademy.com 
www.euclideanspace.com 