Assignment 1

Re-submit Assignment

Due Friday by 18:00 **Points** 10 **Submitting** a file upload **File Types** zip
Available after Mar 1 at 0:00

Overview

There is one assignment in three stages. After the first stage, each builds on the work of the previous one.

The assignment is to build an advanced simulation of **Pool (https://en.wikipedia.org/wiki/Pool_(cue_sports))** , utilising 6 design patterns in 3 stages (2 per stage). The game consists of a rectangular table with several balls on it. If the balls approach the side of the table they will bounce back. Balls also bounce off other balls. The player can hit one of the balls, the white cue ball, with the cue stick. The standard table has 6 pockets into which the balls can fall. The game is single player and is finished when all the balls, other than the cue ball, are in the pockets.

The first stage you will do yourself.

At stages two and three, you will extend someone else's code from your class. Your tutor will determine whose. Make sure your code is nice to maintain.

Assignment 1

Create a simple graphical display using QT.

It must read configuration options from a configuration file "config.json". You can use QJsonDocument to aid decoding.

The layout of this file must be in the specified format.

Your implementation will display the table, a single white ball, and an arbitrary number of other balls.

The balls must be able to bounce off the sides of the table, and each other.

The balls slow down due to friction.

The class you design to set up the game object should utilise the **Abstract Factory** design pattern. This is so that later stages of the assignment can use a different concrete implementation to create the game objects. I.e. you will have an abstract base class and a concrete implementation for stage 1. Your factory will have methods to set up the game such as a `Table* createTable` factory method. You must also use a **Builder** to build some aspect of the game.

Config File

The Configuration file stores:

- The **table** which consists of:
  - the **size**
    - **x**
    - **y**,
  - **colour**,
  - **friction**
- An array of b**alls** each element of which consists of
  - **colour**
  - **mass**
  - **radius**
  - **position**
    - **x**
    - **y**
  - **velocity**
    - **x**
    - **y**

Words in bold are the actual keys that should be used. Colours can be specified by name eg "red' or "blue" if they are simple, or by rgb hex value eg "\#112233". Your program must accept either. All other values are floats.

An example config file is provided.

```
{
"table" : {
 "colour":"green",
 "size":{
 "x":600,
 "y":300
 },
 "friction":0.1
},
"balls" : [
 {"colour":"white","position":{"x":50,"y":50},"velocity":{"x":10,"y":1},"mas
s":1,"radius":10},
 {"colour":"red","position":{"x":150,"y":60},"velocity":{"x":-10,"y":0},"mas
s":1,"radius":10},
 {"colour":"blue","position":{"x":150,"y":40},"velocity":{"x":-10,"y":0},"mas
s":1,"radius":10},
 {"colour":"#123456","position":{"x":250,"y":70},"velocity":{"x":-10,"y":
0},"mass":5,"radius":20}
]
}
```

Physics

Friction

You are free to implement friction any way you want e.g. constant deceleration or deceleration proportional to velocity, as long the balls eventually come to rest.

Collisions with walls

When a ball collides with a wall it should simply reverse the velocity perpendicular to the wall, e.g., if a ball is travelling to the bottom right with a velocity of (1,1) and bounces off the right wall it will be (-1,1), the x velocity is reversed. If it then proceeds to bounce off the bottom wall it becomes (-1,-1) as the y velocity is reversed. Note that in QT, as in many gui environments, the origin is at the the top left and y increases down the screen.

Ball-Ball collisions

Since the collision mechanics between balls of arbitrary mass are non trivial the code for calculating the resulting velocity changes is provided.

You do not need to understand how the mathematical section works (it is somewhat simplified anyway) but you will have to understand the initial properties and then apply the resulting changes in velocities to the respective balls.

```
//QVector2D is a QT class representing a vector in 2D space,
//it has useful functions like dotproduct(), length() and normalize().
//You don't need to use it if you would rather do this yourself
```

```
 //Properties of two colliding balls,
 //ball A
 QVector2D posA;
 QVector2D velA;
 float massA;
 //and ball B
 QVector2D posB;
 QVector2D velB;
 float massB;

 //calculate their mass ratio
 float mR = massB / massA;

 //calculate the axis of collision
 QVector2D collisionVector = posB - posA;
 collisionVector.normalize();

//the proportion of each balls velocity along the axis of collision
 double vA = QVector2D::dotProduct(collisionVector, velA);
 double vB = QVector2D::dotProduct(collisionVector, velB);
 //the balls are moving away from each other so do nothing
 if (vA <= 0 && vB >= 0) {
  return;
 }

 //The velocity of each ball after a collision can be found by solving the quadratic equation
 //given by equating momentum and energy before and after the collision and finding the veloc
ities
 //that satisfy this
 //-(mR+1)x^2 2*(mR*vB+vA)x -((mR-1)*vB^2+2*vA*vB)=0
 //first we find the discriminant
 double a = -(mR + 1);
 double b = 2 * (mR * vB + vA);
 double c = -((mR - 1) * vB * vB + 2 * vA * vB);
 double discriminant = sqrt(b * b - 4 * a * c);
 double root = (-b + discriminant)/(2 * a);
 //only one of the roots is the solution, the other pertains to the current velocities
 if (root - vB < 0.01) {
   root = (-b - discriminant)/(2 * a);
 }


 //The resulting changes in velocity for ball A and B
 QVector2D deltaVA = mR * (vB - root) * collisionVector;
 QVector2D deltaVB = (root - vB) * collisionVector;
```

Marking

There are two segments to this assignment. The code which is worth **7** and the essay which is worth **3**.

The marking (out of 7 marks) for the code section is as follows:

Compiling and running on the lab machines. **(10%)**

Behaviour is read from the config file (e.g. positions, colour of balls, colour and size of table, etc). **(15%)**

The balls bounce off the sides and each other and slow down due to friction. (**15%)**

Appropriate use of design patterns (Use of factory method in creation of game elements and implementation of the builder, and possible bonus marks for other design patterns used). **(30%)**

Sensible / appropriate OO design (Extensible code design, clear separation of responsibility, etc). **(10%)**

Clear code: meaningful variable and method names. **(10%)**

Documentation: Doxygen comments for methods/classes as well as in-code comments where appropriate. **(10%)**

The marking (out of 3 marks) for the essay section is as follows

Explain the application of the design patterns for your concrete code base. **(33%)**

Explain advantage and disadvantages of the design patterns used with respect to your code. **(33%)**

Graphical visualisation in UML of your design patterns. **(33%)**

Academic honesty

While the University is aware that the vast majority of students and staff act ethically and honestly, it is opposed to and will not tolerate academic dishonesty or plagiarism and will treat all allegations of dishonesty seriously.

Further information on academic honesty, academic dishonesty, and the resources available to all students can be found on the academic integrity pages on the current students website: **https://sydney.edu.au/students/academic-integrity.html (https://sydney.edu.au/students/academic-integrity.html)** .

Compliance statement

In submitting this work, I acknowledge I have understood the following:

- I have read and understood the University of Sydney's **Academic Honesty in Coursework Policy 2015 (https://sydney.edu.au/policies/showdoc.aspx?recnum=PDOC2012/254&RendNum=0)** .
- The work is substantially my own and where any parts of this work are not my own I have indicated this by acknowledging the source of those parts of the work and enclosed any quoted text in quotation marks.
- The work has not previously been submitted in part or in full for assessment in another unit unless I have been given permission by my unit of study coordinator to do so.
- The work will be submitted to similarity detection software (Turnitin) and a copy of the work will be retained in Turnitin's paper repository for future similarity checking.
- Engaging in plagiarism or academic dishonesty will, if detected, lead to the University commencing proceedings under the **Academic Honesty in Coursework Policy 2015 (https://sydney.edu.au/policies/showdoc.aspx?recnum=PDOC2012/254&RendNum=0)** and the **Academic Honesty Procedures 2016 (http://sydney.edu.au/policies/default.aspx?mode=glossary&word=Academic+honesty)** .
- Engaging another person to complete part or all of the submitted work will, if detected, lead to the University commencing proceedings against me for potential student misconduct under the **University of Sydney (Student Discipline) Rule 2016 (http://sydney.edu.au/policies/showdoc.aspx?recnum=PDOC2017/441&RendNum=0)** .