Object Oriented Design Patterns

Assignment 2

Kashif Akram – 460458380

Received Code:

1. Code Design

Code was designed very well for extension, proper implementation of factory and builder design patterns. SOLID principles were well implemented as well, each class has its own purpose and great cohesion between parent and child classes.

In my point of view developer of this program has extensive c++ syntax and library knowledge as he already implemented unique pointers and made use of util class even before discussed in lectures or tutorials.

2. Hindered Elements

Well it's not really hindering part, but we can say some minor mistake in code implementation as instead of writing getters and setters of table and balls member variables those variables were declared as public and external classes had direct access to member variables.

3. Documentation

I didn't receive external documentation about this code, so no idea about that but I found internal documentation is written for every method, very concise and easily understandable.

4. Code Implementation

As I mentioned in code design section, coding is done professionally and easy to extend with use of c++ 11 features and syntax which are even not discussed in lectures or tutorials.

This code is implemented in very hierarchical way which makes is easy to understand and extend.

I would have separated file operations in separate class having all functions to manipulate external files instead of writing code in main.cpp so that in future if we need to extend code to read config and may be write log from more than one files I would just need to call my existing functions without writing any further code.

5. **Code Style**

I think files, variable, class, function names are consistent and make sense, you can understand the purpose of each of them by their names.

Code was written as if it was written by professional and experienced developer, I mean if same code was to be written by newbie he/she would have write 2x more coding lines and would have many unused functions and variables.

My Code:

1. **Adapter**

Application of design patterns in the project:

I used (Stage1Engine, UniversalEngine) adapter to use my own stage 1 engine functions to be used and extended in stage 2 engine.

Advantages:

- Code reuse, no need to write new code from scratch
- Adapter can be implemented to use legacy code in whatever style, format it was written
- Used to create compatibility between old and new incompatible interfaces

Disadvantages:

- May a little performance issue as all the requests will be forwarded through adapter

- Might need whole a lot of adapters chain to reach the required type

2. Composite

Application in the project:

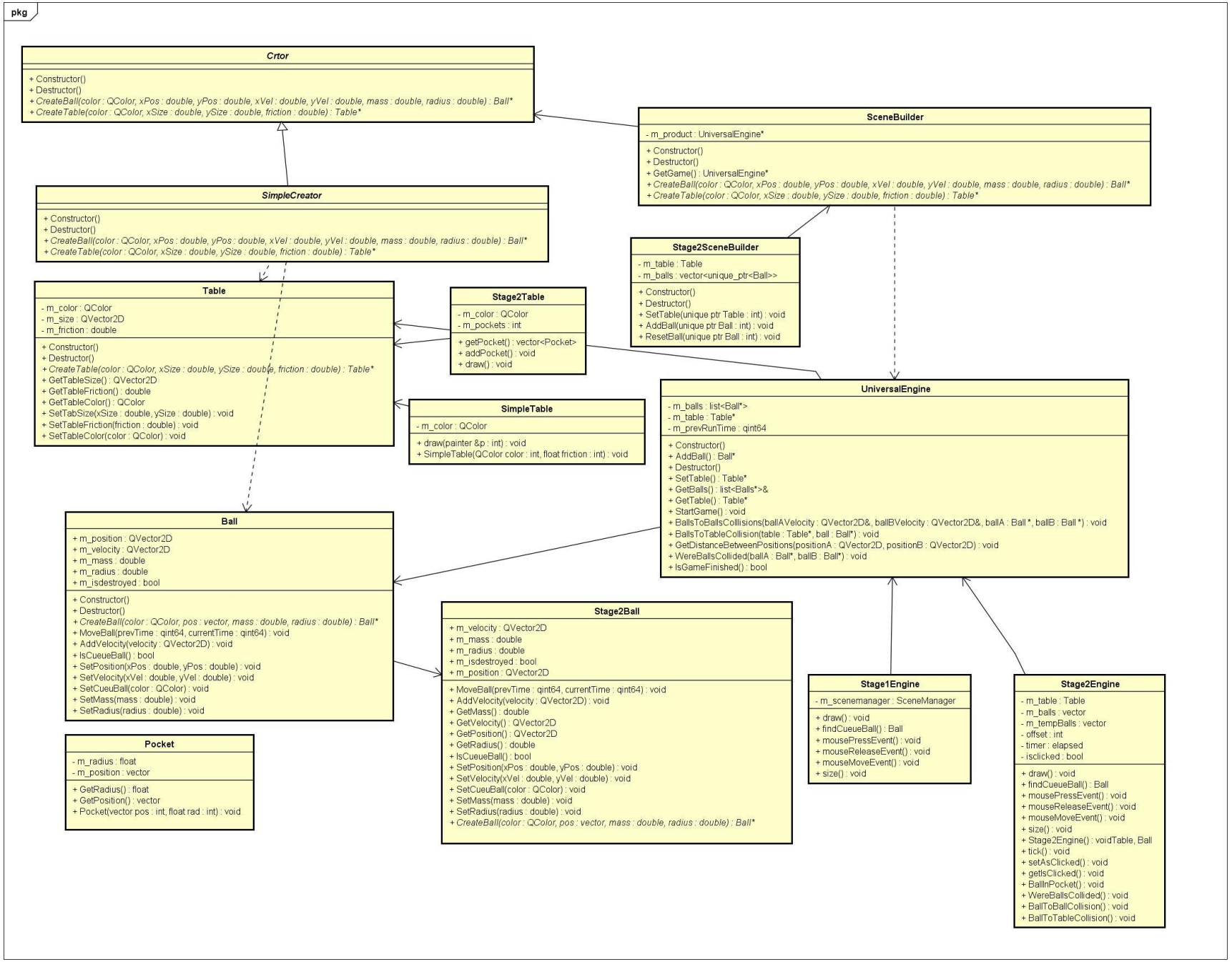
2D vector of balls in stage 2 concrete balls class is used as composite design to define child balls in parent.

Advantages:

- Composite design pattern offers tree structure implementation of parent child objects
- Objects by themselves and their existence in aggregation of objects may have same implementation and will be handled in same way.
- Very useful to implement hierarchies of objects to allow client a uniform way to implement each object in hierarchy.

Disadvantages:

- Limited in a way that it's easy to add any type of components which makes it difficult to apply constraints on specific type of components to be added.
- Resultant design looks very broad and abstract.


pkg
Crtor
+ Constructor()
+ Destructor()
+ CreateBall(color : QColor, xPos : double, yPos : double, xVel : double, yVel : double, mass : double, radius : double) : Ball*
+ CreateTable(color : QColor, xSize : double, ySize : double, friction : double) : Table*
SceneBuilder
- m_product : UniversalEngine*
+ Constructor()
+ Destructor()
+ GetGame() : UniversalEngine*
+ CreateBall(color : QColor, xPos : double, yPos : double, xVel : double, yVel : double, mass : double, radius : double) : Ball*
+ CreateTable(color : QColor, xSize : double, ySize : double, friction : double) : Table*
SimpleCreator
+ Constructor()
+ Destructor()
+ CreateBall(color : QColor, xPos : double, yPos : double, xVel : double, yVel : double, mass : double, radius : double) : Ball*
+ CreateTable(color : QColor, xSize : double, ySize : double, friction : double) : Table*
Stage2SceneBuilder
- m_table : Table
- m_balls : vector<unique_ptr<Ball>>
+ Constructor()
+ Destructor()
+ SetTable(unique ptr Table : int) : void
+ AddBall(unique ptr Ball : int) : void
+ ResetBall(unique ptr Ball : int) : void
Table
- m_color : QColor
- m_size : QVector2D
- m_friction : double
+ Constructor()
+ Destructor()
+ CreateTable(color : QColor, xSize : double, ySize : double, friction : double) : Table*
+ GetTableSize() : QVector2D
+ GetTableFriction() : double
+ GetTableColor() : QColor
+ SetTabSize(xSize : double, ySize : double) : void
+ SetTableFriction(friction : double) : void
+ SetTableColor(color : QColor) : void
Stage2Table
- m_color : QColor
- m_pockets : int
+ getPocket() : vector<Pocket>
+ addPocket() : void
+ draw() : void
SimpleTable
- m_color : QColor
+ draw(painter &p : int) : void
+ SimpleTable(QColor color : int, float friction : int) : void
UniversalEngine
- m_balls : list<Ball*>
- m_table : Table*
- m_prevRunTime : qint64
+ Constructor()
+ AddBall() : Ball*
+ Destructor()
+ SetTable() : Table*
+ GetBalls() : list<Balls*>&
+ GetTable() : Table*
+ StartGame() : void
+ BallsToBallsColllisions(ballAVelocity : QVector2D&, ballBVelocity : QVector2D&, ballA : Ball *, ballB : Ball *) : void
+ BallsToTableCollision(table : Table*, ball : Ball*) : void
+ GetDistanceBetweenPositions(positionA : QVector2D, positionB : QVector2D) : void
+ WereBallsCollided(ballA : Ball*, ballB : Ball*) : void
+ IsGameFinished() : bool
Ball
+ m_position : QVector2D
+ m_velocity : QVector2D
+ m_mass : double
+ m_radius : double
+ m_isdestroyed : bool
+ Constructor()
+ Destructor()
+ CreateBall(color : QColor, pos : vector, mass : double, radius : double) : Ball*
+ MoveBall(prevTime : qint64, currentTime : qint64) : void
+ AddVelocity(velocity : QVector2D) : void
+ IsCueueBall() : bool
+ SetPosition(xPos : double, yPos : double) : void
+ SetVelocity(xVel : double, yVel : double) : void
+ SetCueuBall(color : QColor) : void
+ SetMass(mass : double) : void
+ SetRadius(radius : double) : void
Stage2Ball
+ m_velocity : QVector2D
+ m_mass : double
+ m_radius : double
+ m_isdestroyed : bool
+ m_position : QVector2D
+ MoveBall(prevTime : qint64, currentTime : qint64) : void
+ AddVelocity(velocity : QVector2D) : void
+ GetMass() : double
+ GetVelocity() : QVector2D
+ GetPosition() : QVector2D
+ GetRadius() : double
+ IsCueueBall() : bool
+ SetPosition(xPos : double, yPos : double) : void
+ SetVelocity(xVel : double, yVel : double) : void
+ SetCueuBall(color : QColor) : void
+ SetMass(mass : double) : void
+ SetRadius(radius : double) : void
+ CreateBall(color : QColor, pos : vector, mass : double, radius : double) : Ball*
Pocket
- m_radius : float
- m_position : vector
+ GetRadius() : float
+ GetPosition() : vector
+ Pocket(vector pos : int, float rad : int) : void
Stage1Engine
- m_scenemanager : SceneManager
+ draw() : void
+ findCueueBall() : Ball
+ mousePressEvent() : void
+ mouseReleaseEvent() : void
+ mouseMoveEvent() : void
+ size() : void
Stage2Engine
- m_table : Table
- m_balls : vector
- m_tempBalls : vector
- offset : int
- timer : elapsed
- isclicked : bool
+ draw() : void
+ findCueueBall() : Ball
+ mousePressEvent() : void
+ mouseReleaseEvent() : void
+ mouseMoveEvent() : void
+ size() : void
+ Stage2Engine() : voidTable, Ball
+ tick() : void
+ setAsClicked() : void
+ getIsClicked() : void
+ BallInPocket() : void
+ WereBallsCollided() : void
+ BallToBallCollision() : void
+ BallToTableCollision() : void