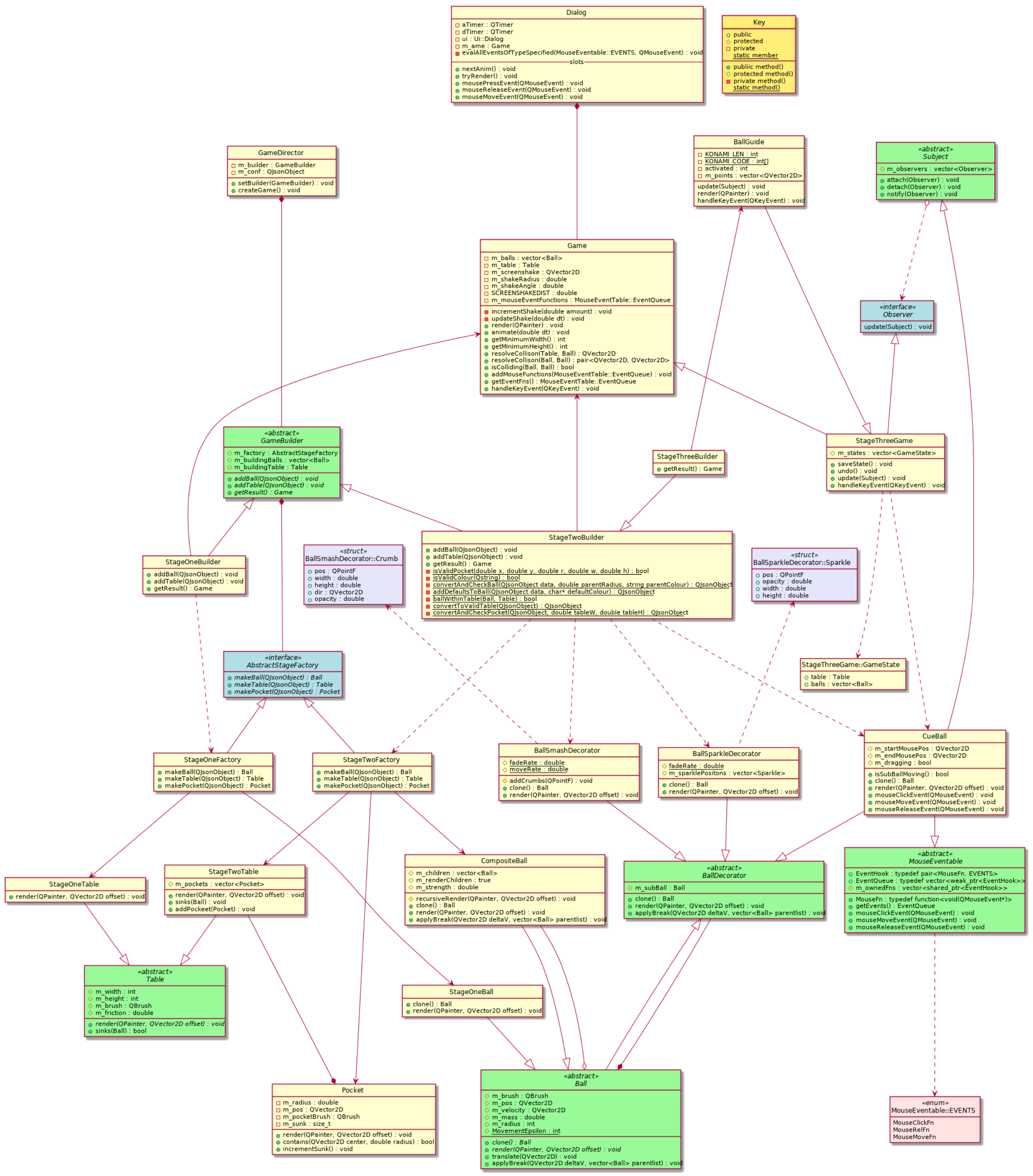
PoolGame v3

enter Konami code to activate extension features

The received code was not designed for extensibility. It did not apply interface segregation or dependency inversion, leading to an overpopulated interface often with non-virtual methods that do nothing. This is caused in particular by the Game class being shared between state1 and 2 implementation, even though it was clearly not written for that purpose. Another point of contention is the 'magic' MouseEventAble class, which couples its subclass with itself and dialog. It also makes it awkward to try and handle key events through a different interface, or to expand the interface to include key events. Implementation wise, there were a few bugs and some sections where drawing and animation is coupled when it appears explicitly seperate to the client class. There were missing documentation on several classes, but the most severe offender is the MoseEventAble class which tells the reader to 'go away'. The code style was inconsistent - some constants were capitalised, some member variables did not follow the 'm_name' convention.

To implement the undo functionality, the memento and prototype pattern was used. The memento allows the state of the game to be stored and restored, while the prototype allows objects to be created in the same state as the prototype. The memento pattern can lead to high memory usage and leaks if not implemented carefully. With respect to the code base, where states are often stored in a pointer, copying and moving data can cause memory fragmentation, slowing down the program. A drawback of the prototype pattern is that is pollutes the class heirarchy the clone method, making it cluttered and obscuring other important functionality.

The Observer pattern was used for the undo function and the extension. By having the CueBall class notify its observers of successful mouse fire events, we were able to find out when something interesting happens on the CueBall (e.g., a move that should be stored on the caretaker of the memento) without maintaining an explicit reference to the object or inheriting the MouseEventAble class. The current implementation has the limitation that the observer cannot select which event to observe and information about the event can only be found from a reference to the subject passed in the update method. A common concern of this pattern is that the update method can be expensive and cause more updates. Given that the project is in its final stage and there's only one subject and two observers, this is not a major concern in the code base.


Dialog
aTimer : QTimer
dTimer : QTimer
ui : Ui::Dialog
m_ame : Game
evalAllEventsOfTypeSpecified(MouseEventable::EVENTS, QMouseEvent) : void
slots
nextAnim() : void
tryRender() : void
mousePressEvent(QMouseEvent) : void
mouseReleaseEvent(QMouseEvent) : void
mouseMoveEvent(QMouseEvent) : void
Key
public
protected
private
static member
publiic method()
protected method()
private method()
static method()
GameDirector
m_builder : GameBuilder
m_conf : QJsonObject
setBuilder(GameBuilder) : void
createGame() : void
BallGuide
KONAMI_LEN : int
KONAMI_CODE : int[]
activated : int
m_points : vector<QVector2D>
update(Subject) : void
render(QPainter) : void
handleKeyEvent(QKeyEvent) : void
«abstract»
Subject
m_observers : vector<Observer>
attach(Observer) : void
detach(Observer) : void
notify(Observer) : void
Game
m_balls : vector<Ball>
m_table : Table
m_screenshake : QVector2D
m_shakeRadius : double
m_shakeAngle : double
SCREENSHAKEDIST : double
m_mouseEventFunctions : MouseEventTable::EventQueue
incrementShake(double amount) : void
updateShake(double dt) : void
render(QPainter) : void
animate(double dt) : void
getMinimumWidth() : int
getMinimumHeight() : int
resolveCollison(Table, Ball) : QVector2D
resolveCollison(Ball, Ball) : pair<QVector2D, QVector2D>
isColliding(Ball, Ball) : bool
addMouseFunctions(MouseEventTable::EventQueue) : void
getEventFns() : MouseEventTable::EventQueue
handleKeyEvent(QKeyEvent) : void
«interface»
Observer
update(Subject) : void
«abstract»
GameBuilder
m_factory : AbstractStageFactory
m_buildingBalls : vector<Ball>
m_buildingTable : Table
addBall(QJsonObject) : void
addTable(QJsonObject) : void
getResult() : Game
StageThreeBuilder
getResult() : Game
StageThreeGame
m_states : vector<GameState>
saveState() : void
undo() : void
update(Subject) : void
handleKeyEvent(QKeyEvent) : void
StageOneBuilder
addBall(QJsonObject) : void
addTable(QJsonObject) : void
getResult() : Game
«struct»
BallSmashDecorator::Crumb
pos : QPointF
width : double
height : double
dir : QVector2D
opacity : double
StageTwoBuilder
addBall(QJsonObject) : void
addTable(QJsonObject) : void
getResult() : Game
isValidPocket(double x, double y, double r, double w, double h) : bool
isValidColour(Qstring) : bool
convertAndCheckBall(QJsonObject data, double parentRadius, string parentColour) : QJsonObject
addDefaultsToBall(QJsonObject data, char* defaultColour) : QJsonObject
ballWithinTable(Ball, Table) : bool
convertToValidTable(QJsonObject) : QJsonObject
convertAndCheckPocket(QJsonObject, double tableW, double tableH) : QJsonObject
«struct»
BallSparkleDecorator::Sparkle
pos : QPointF
opacity : double
width : double
height : double
«interface»
AbstractStageFactory
makeBall(QJsonObject) : Ball
makeTable(QJsonObject) : Table
makePocket(QJsonObject) : Pocket
StageThreeGame::GameState
table : Table
balls : vector<Ball>
StageOneFactory
makeBall(QJsonObject) : Ball
makeTable(QJsonObject) : Table
makePocket(QJsonObject) : Pocket
StageTwoFactory
makeBall(QJsonObject) : Ball
makeTable(QJsonObject) : Table
makePocket(QJsonObject) : Pocket
BallSmashDecorator
fadeRate : double
moveRate : double
addCrumbs(QPointF) : void
clone() : Ball
render(QPainter, QVector2D offset) : void
BallSparkleDecorator
fadeRate : double
m_sparklePositons : vector<Sparkle>
clone() : Ball
render(QPainter, QVector2D offset) : void
CueBall
m_startMousePos : QVector2D
m_endMousePos : QVector2D
m_dragging : bool
isSubBallMoving() : bool
clone() : Ball
render(QPainter, QVector2D offset) : void
mouseClickEvent(QMouseEvent) : void
mouseMoveEvent(QMouseEvent) : void
mouseReleaseEvent(QMouseEvent) : void
CompositeBall
m_children : vector<Ball>
m_renderChildren : true
m_strength : double
recursiveRender(QPainter, QVector2D offset) : void
clone() : Ball
render(QPainter, QVector2D offset) : void
applyBreak(QVector2D deltaV, vector<Ball> parentlist) : void
«abstract»
BallDecorator
m_subBall : Ball
clone() : Ball
render(QPainter, QVector2D offset) : void
applyBreak(QVector2D deltaV, vector<Ball> parentlist) : void
«abstract»
MouseEventable
EventHook : typedef pair<MouseFn, EVENTS>
EventQueue : typedef vector<weak_ptr<EventHook>>
m_ownedFns : vector<shared_ptr<EventHook>>
MouseFn : typedef function<void(QMouseEvent*)>
getEvents() : EventQueue
mouseClickEvent(QMouseEvent) : void
mouseMoveEvent(QMouseEvent) : void
mouseReleaseEvent(QMouseEvent) : void
StageOneTable
render(QPainter, QVector2D offset) : void
StageTwoTable
m_pockets : vector<Pocket>
render(QPainter, QVector2D offset) : void
sinks(Ball) : void
addPockeet(Pocket) : void
«abstract»
Table
m_width : int
m_height : int
m_brush : QBrush
m_friction : double
render(QPainter, QVector2D offset) : void
sinks(Ball) : bool
StageOneBall
clone() : Ball
render(QPainter, QVector2D offset) : void
Pocket
m_radius : double
m_pos : QVector2D
m_pocketBrush : QBrush
m_sunk : size_t
render(QPainter, QVector2D offset) : void
contains(QVector2D center, double radius) : bool
incrementSunk() : void
«abstract»
Ball
m_brush : QBrush
m_pos : QVector2D
m_velocity : QVector2D
m_mass : double
m_radius : int
MovementEpsilon : int
clone() : Ball
render(QPainter, QVector2D offset) : void
translate(QVector2D) : void
applyBreak(QVector2D deltaV, vector<Ball> parentlist) : void
«enum»
MouseEventable::EVENTS
MouseClickFn
MouseRelFn
MouseMoveFn