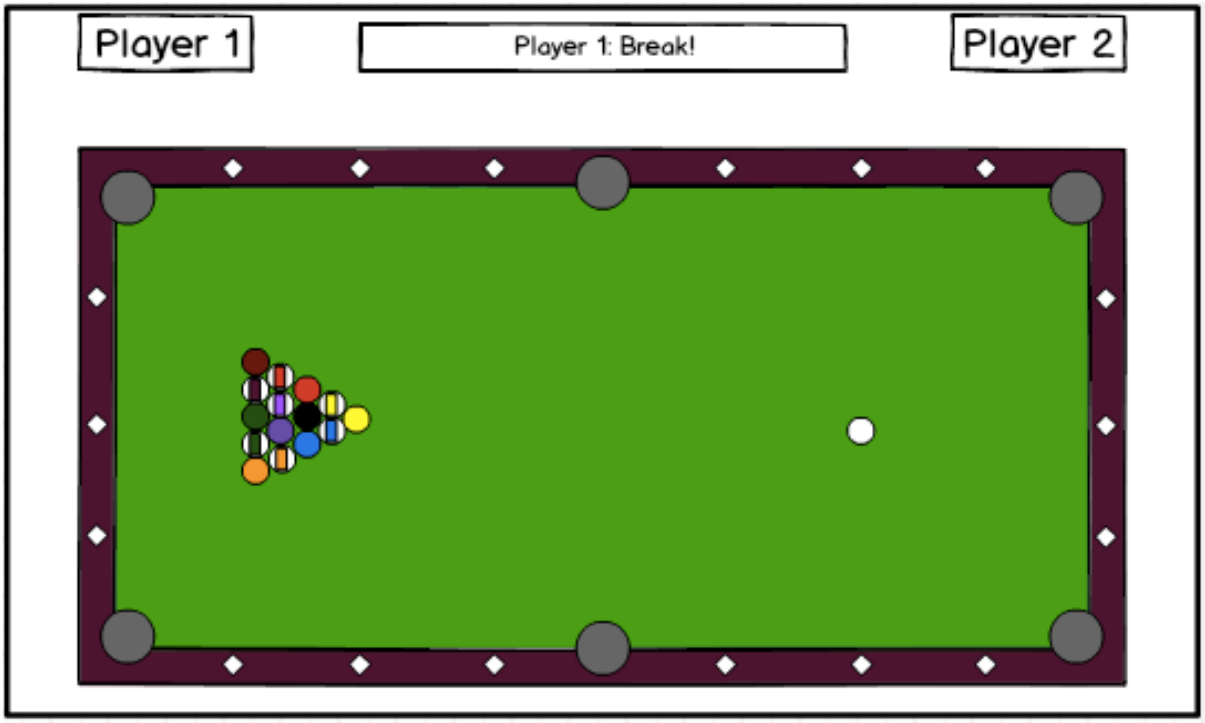
Pool

Brief Description:
We are implementing a two-team turn based 8-ball pool game. Each team can have an unlimited number of human players. Each player can control the direction and power of the cue stick, and in turn, the direction and power with which the balls are hit. The goal of the game is to be the first to pocket all of either the striped or solid balls and then pocket the black 8-ball. The team that does that first wins.

Feature List

1. We have a main top-down view of a pool table with balls placed in initial formation.
2. Players adjust the pool stick around the cue ball to set cue direction.
3. Players move cursor farther away from the cue ball or hold down the spacebar to increase intensity.
4. Game offers realistic physical collisions for stick to ball, ball to ball, and ball to rail.
5. Players see their pocketed balls under their name.
6. Players see the remaining balls they must hit under their names.
7. Players have the option to set timer for turns.
8. Players lag at the beginning of the game to determine who goes first.
9. Players choose to play against a computer or against friends.
10. Players set the difficulty of a computer player.
11. Players select between two game modes, 8-ball pool and 9-ball pool.
12. Players choose under game options if they want to see the path line of the cue to the ball, path line of the ball after being hit, or direction that the ball will travel in.



Storyboard

Interfaces of major software components

We will use a model-view-controller design to implement our game. Our controller will detect mouse presses and position to be passed to the model, which will subsequently affect the view.

MODEL

- class Ball (subclass CueBall, subclass NumberBall)
    - Instance variables:
        - Point position (x_pos, y_pos) - coordinates of the ball on the table
        - boolean isPocketed: whether the ball has been pocketed
        - double angle
        - double speed
        - double radius
        - static final int id: number on side of ball, 0 for cue ball
    - constructor: Ball(int id) -- number on side of ball if NumberBall, 0 if CueBall
    - updateVelocity() - changes speed and direction if speed > 0
    - subclass NumberBall extends Ball
        - constructor: NumberBall(int id)
    - subclass CueBall extends Ball
        - constructor: CueBall(Point loc)
        - scratchMove()
- class GameBoard
    - Instance Variables:

        - Ball[16] balls
        - Boundary[4] boundaries
        - Pocket[6] pockets
    - constructor: GameBoard()
    - Methods
        - move(): changes position of balls on the table, loops through collisions to check if any collisions occur (loops through three object collisions, then two object collisions), update speed and angles of balls involved in collisions
- class Boundary
    - Instance Variables:
        - Point[2] vertices - two points defining the boundary line
    - constructor: Boundary(Point a, Point b)
- class Pocket
    - Instance Variables:
        - double radius: radius of pocket
        - Point location: location of pockets
    - constructor: Pocket(Point loc)
- class CueStick
    - Instance variables:
        - location (x_pos, y_pos): location of tip of cue stick
        - double angle
        - double speed
    - constructor: CueStick()
- class Team
    - Instance variables:
        - int teamBallType: 1 if striped, 0 if solid, null otherwise (if not set yet)
        - ArrayList<Ball> pocketedBalls
        - ArrayList<Ball> remainingBalls
        - canPocketEightBall
    - constructor: Team()
- class Collisions
    - Instance variables:
        - Object obj1
        - Object obj2
        - Object obj3
    - Constructor: Collision(obj1, obj2)
    - Constructor: Collision(obj1, obj2, obj3)
    - Methods
        - boolean inContact(): if obj1 and obj2 (or obj3) are in the positions to collide (if three objects, all positions must be the same, then return true; if not, returns false.
        - void updateVelocities(): changes the direction and speed of obj1, obj2, and/or obj3 based on momentum laws
- class GameTimer
    - Instance variables
        - Timer timer
- class PoolGame
    - Instance variables

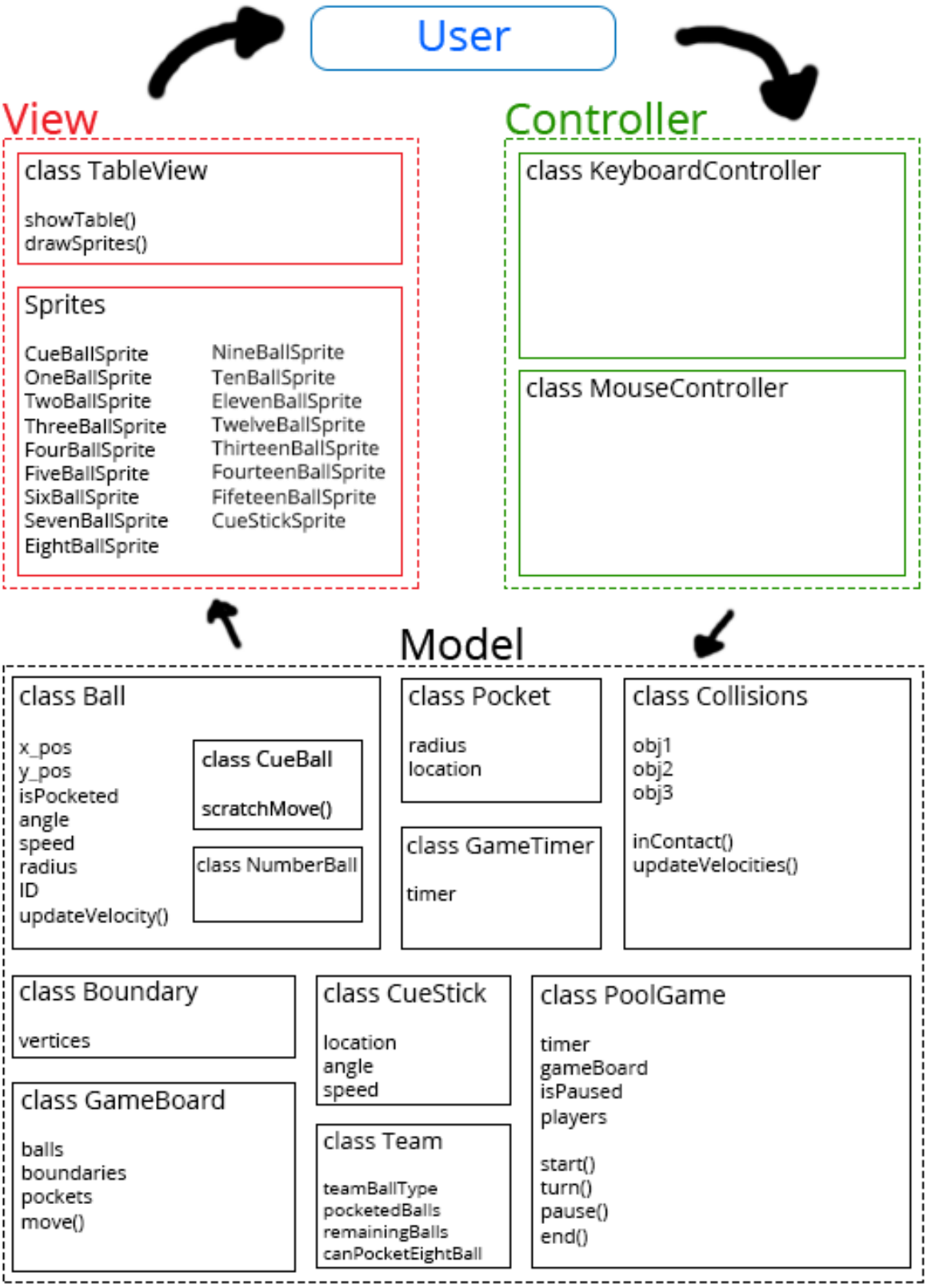
        - GameTimer timer
        - Gameboard gameBoard
        - boolean isPaused
        - Player[] players
    - Methods
        - start(): the main game loop that will run, gets information from the controllers to update the view, calls turn() until game ends
        - turn(): start timer, place CueBall if scratched, set CueBallStick instance variables, make a hit, allow the balls on the board to settle into stable position (call GameBoard.move()), update pocketed balls for each team, switch team if necessary, stop timer after time limit
- pause(): pauses entire game
        - end():

VIEW

- CueBallSprite
- OneBallSprite
- TwoBallSprite
- ThreeBallSprite
- FourBallSprite
- FiveBallSprite
- SixBallSprite
- SevenBallSprite
- EightBallSprite
- NineBallSprite
- TenBallSprite
- ElevenBallSprite
- TwelveBallSprite
- ThirteenBallSprite
- FourteenBallSprite
- FifteenBallSprite
- CueStickSprite
- TableBoardView
    - showTable(): shows the table when the game starts
    - drawSprites(): constantly checks on individual sprites and updates the view to reflect their positions

CONTROLLER

- KeyboardController
- MouseController


User
View
class TableView
showTable()
drawSprites()
Sprites
CueBallSprite
OneBallSprite
TwoBallSprite
ThreeBallSprite
FourBallSprite
FiveBallSprite
SixBallSprite
SevenBallSprite
EightBallSprite
NineBallSprite
TenBallSprite
ElevenBallSprite
TwelveBallSprite
ThirteenBallSprite
FourteenBallSprite
FifeteenBallSprite
CueStickSprite
Controller
class KeyboardController
class MouseController
Model
class Ball
x_pos
y_pos
isPocketed
angle
speed
radius
ID
updateVelocity()
class CueBall
scratchMove()
class NumberBall
class Pocket
radius
location
class GameTimer
timer
class Collisions
obj1
obj2
obj3
inContact()
updateVelocities()
class Boundary
vertices
class GameBoard
balls
boundaries
pockets
move()
class CueStick
location
angle
speed
class Team
teamBallType
pocketedBalls
remainingBalls
canPocketEightBall
class PoolGame
timer
gameBoard
isPaused
players
start()
turn()
pause()
end()

Tests

Ball Class
- constructor: Ball(int id)
  - should assign inputted ID to the ball
  - should throw when input ID is out of range (not 0-15)
- updateVelocity()
  - should correctly update speed and angle

GameBoard Class
- constructor: GameBoard()
  - should assign default values to the gameboard
- move()
  - should correctly change position of balls on the table involved in collisions

Boundary Class
- constructor: Boundary(Point a, Point b)
  - should assign inputted points to the boundary
  - should throw if point is out of bounds

Pocket Class
- constructor: Pocket(Point loc)
  - should assign inputted point to the pocket
  - should throw if point is out of bounds

CueStick Class
- constructor: CueStick()
  - should assign default values to the cue stick

Team Class
- constructor: Team()
  - should assign default values to the team

Collisions Class
- constructor: Collision(obj1, obj2)
  - should assign objects to the collision
- Constructor: Collision(obj1, obj2, obj3)
  - should assign objects to the collision
- boolean inContact()
  - should return true if obj1 and obj2 and/or obj3 are in the positions to collide, should return false if not
  - Specific Test Cases
    - ObjectsWithSamePositionCollideAndShouldReturnTrue()
      Ball b1 = new Ball(1);
      b1.setPosition(50, 50);
      Ball b2 = new Ball(2);
      b2.setPosition(50, 50);
      Collision c = new Collision(b1, b2);
      boolean expected = true;

AssertEquals("Objects with same position did not collide", expected,
c.inContact();)

- TwoObjectsWithDifferentPositionsDoNotCollideAndShouldReturnFalse()
  Ball b1 = new Ball(1);
  b1.setPosition(50, 50);
  Ball b2 = new Ball(2);
  b2.setPosition(40,50);
  Collision c = new Collision(b1, b2);
  boolean expected = false;
  AssertEquals("Two objects with different positions collided", expected,
  c.inContact();)
- ThreeObjectsWithNotSamePositionsDoNotCollideAndShouldReturnFalse()
  Ball b1 = new Ball(1);
  b1.setPosition(50, 50);
  Ball b2 = new Ball(2);
  b2.setPosition(40,50);
  Ball b3 = new Ball(3);
  b3.setPosition(40,50);
  Collision c = new Collision(b1, b2, b3);
  boolean expected = false;
  AssertEquals("Three objects not in same positions collided", expected,
  c.inContact();)

- should update these test cases for objects other than balls and different radii

- void updateVelocities()
  - should correctly change the direction and speed of obj1, obj2, and/or obj3 based on momentum laws