Pool

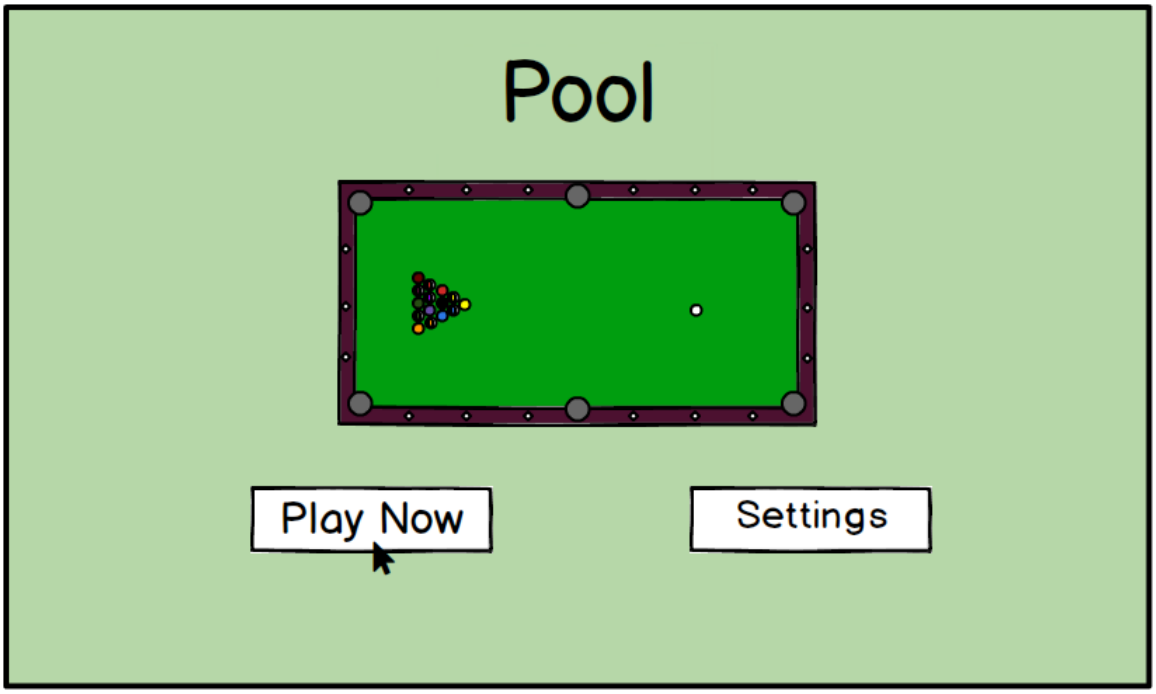
Brief Description:

We are implementing a two-player turn based 8-ball pool game. Each player can control the direction and power of the cue stick to hit the balls on the pool table. 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 player who does that first wins.

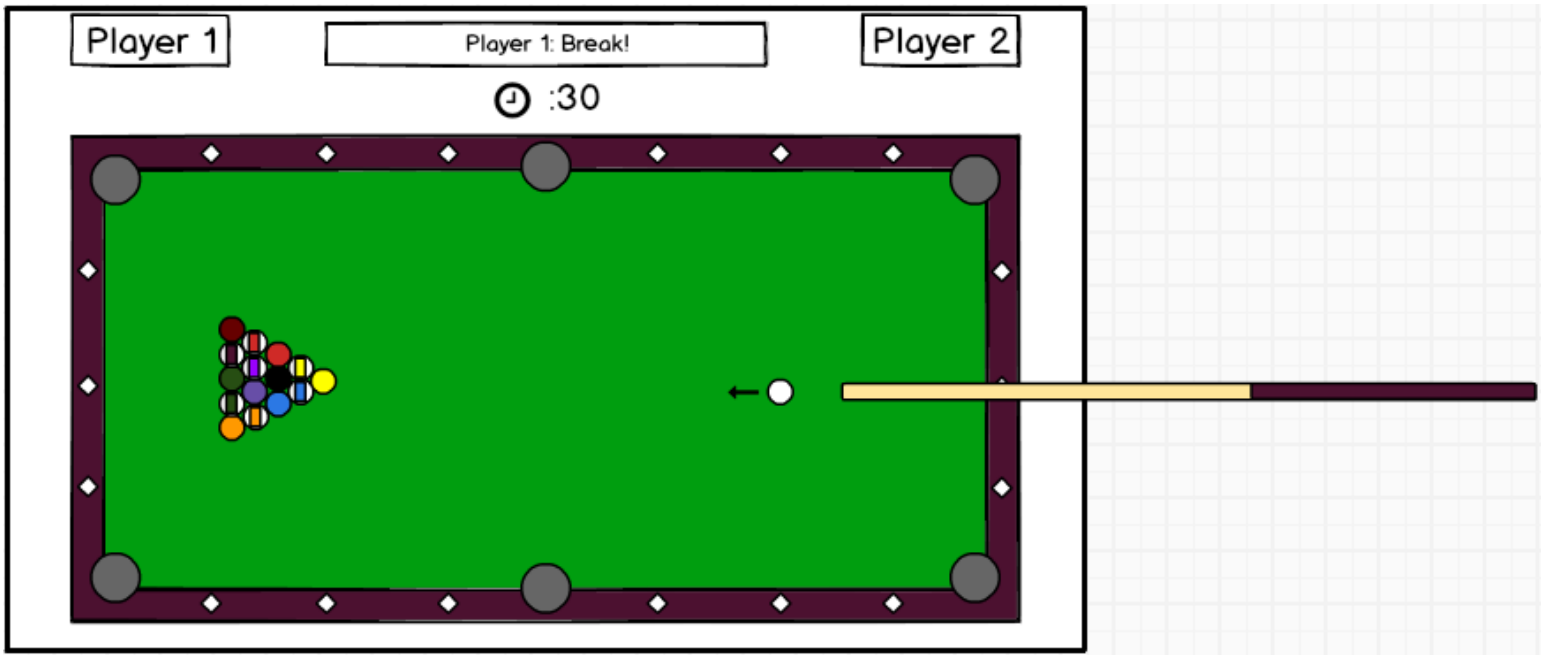
Feature List

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 hold down and move the cursor farther away from the cue ball to adjust cue velocity.
4. Upon release, the cue stick is activated.
5. Game offers realistic physical collisions for stick to ball, ball to ball, and ball to rail.
6. Players see their pocketed balls under their name.
7. Players have the option to set timer for turns.
8. Players choose under game options if they want to see the path line of the ball after being hit, and/or only the direction that the ball will travel in.
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.

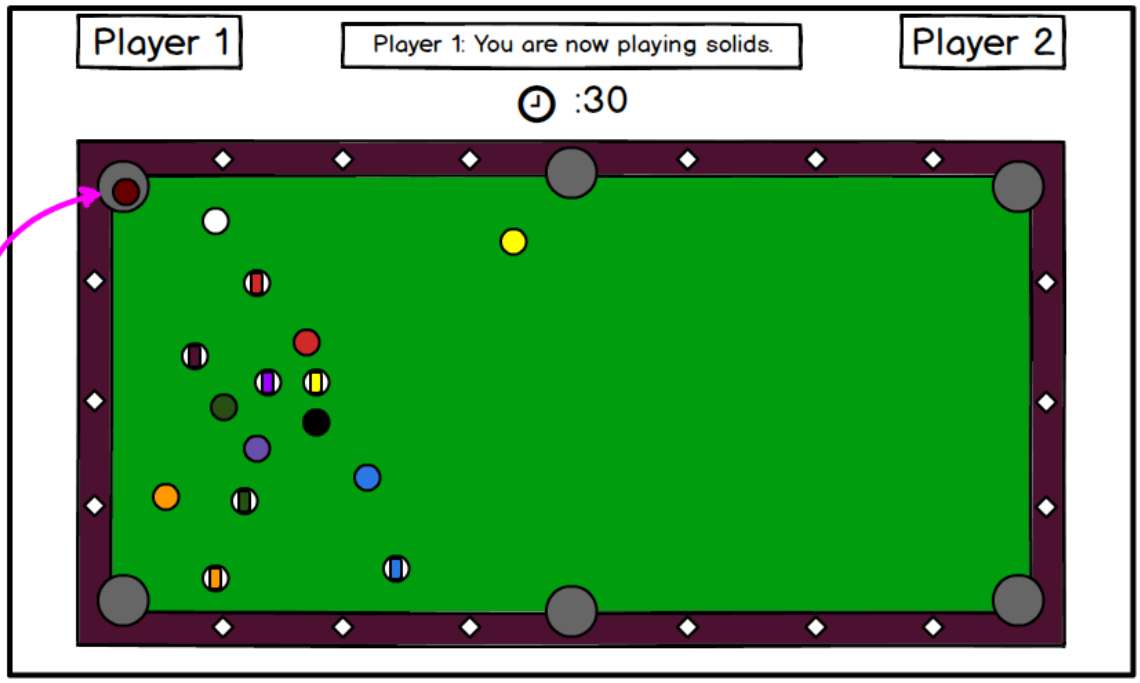
Storyboard



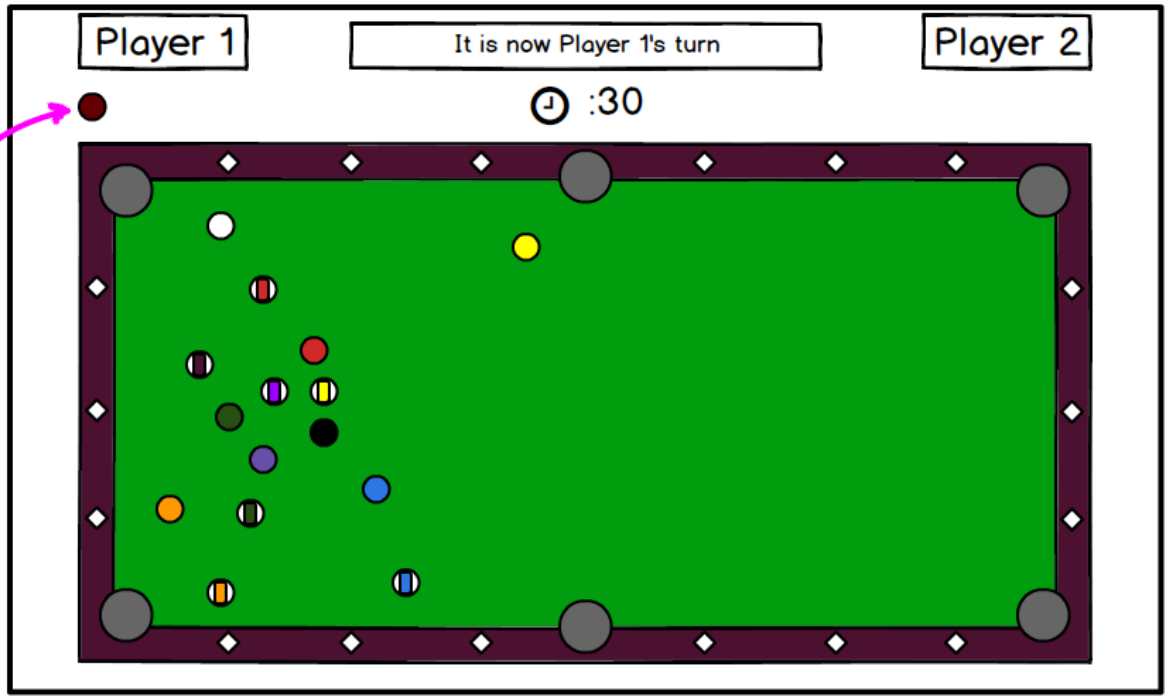
Two players want to play a game of 8-Ball Pool. At the start of the game, a main menu lets them choose whether to play right away or choose settings like having a timer or increasing the difficulty. They click Play Now to continue with default settings.



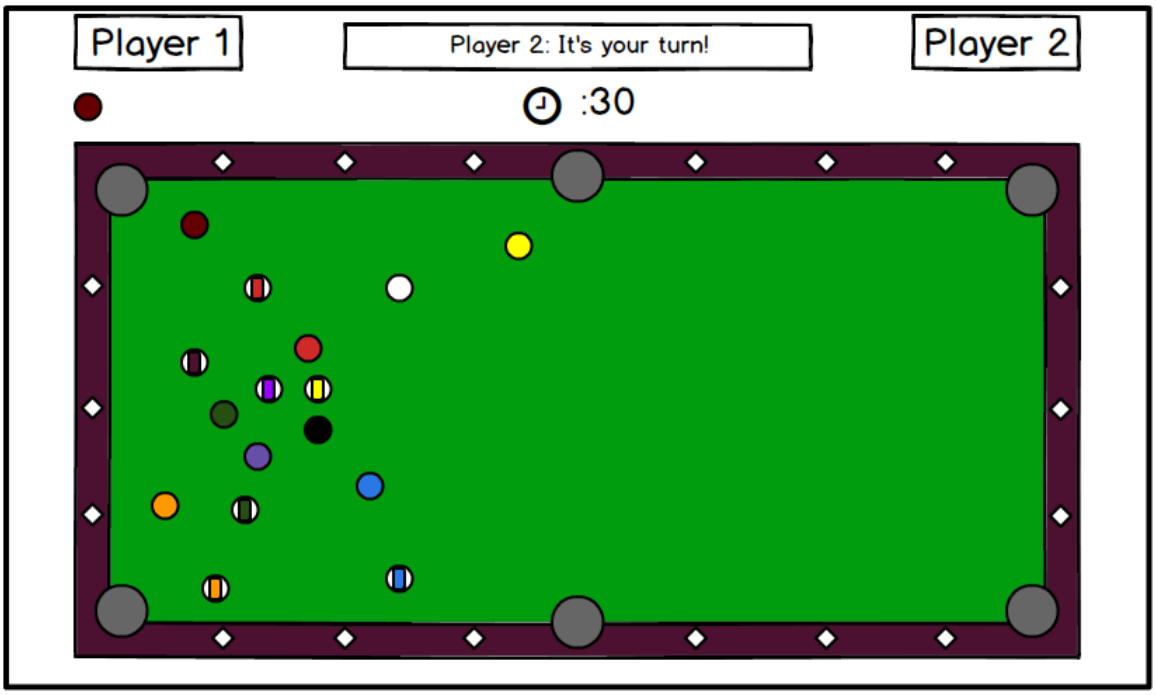
Player 1 begins the game by hitting the cue ball to break the triangle of numbered balls. Player 1's goal is to pocket a ball, which means to hit a ball into one of the pockets.



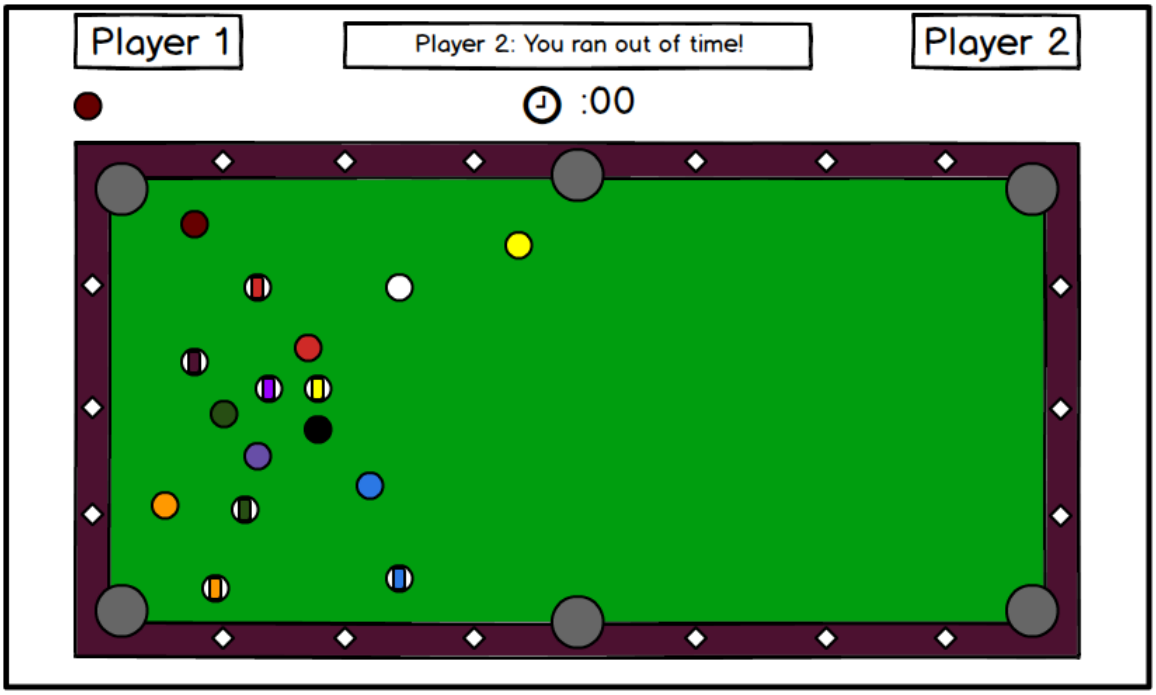
Player 1 pockets a solid ball, so he must continue to play solids in this game. His goal is to pocket all the solids in the game without pocketing the black 8-ball.



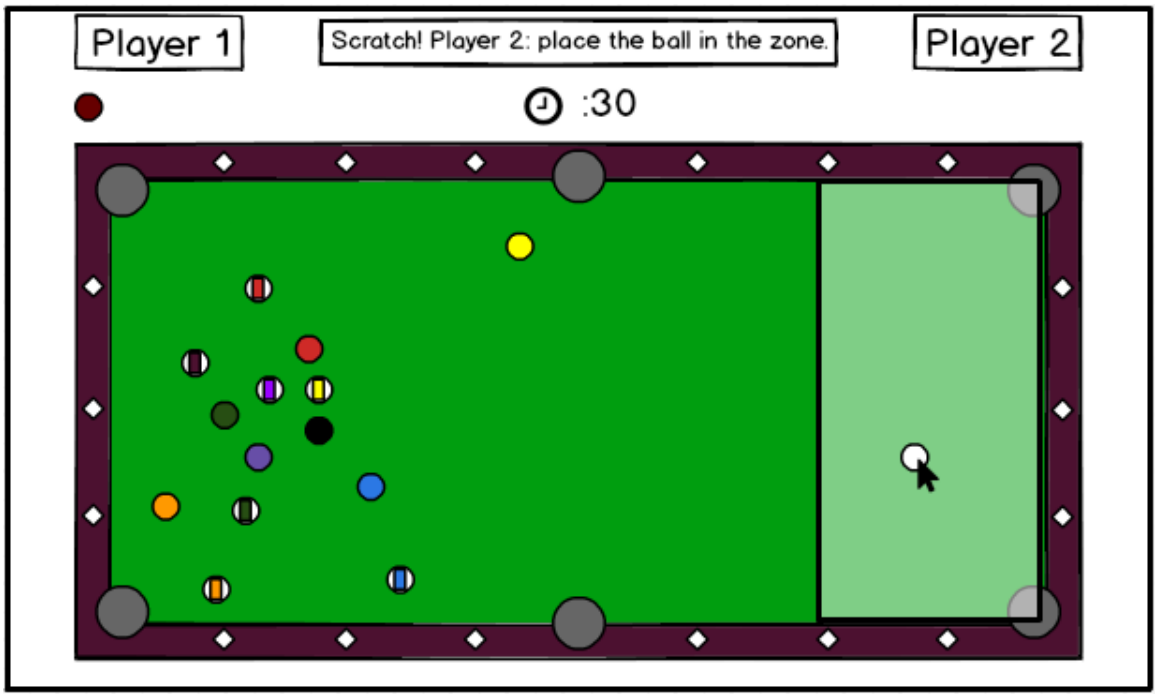
Player 1's pocketed ball appears under his name. Because Player 1 pocketed a ball, he gets to go again. The timer resets to 30 seconds. Player 1 adjusts the intensity and angle of the cue stick by moving the mouse and holding the mouse down. He hits the cue ball. Player 1 continues until he does not pocket a ball on a turn.



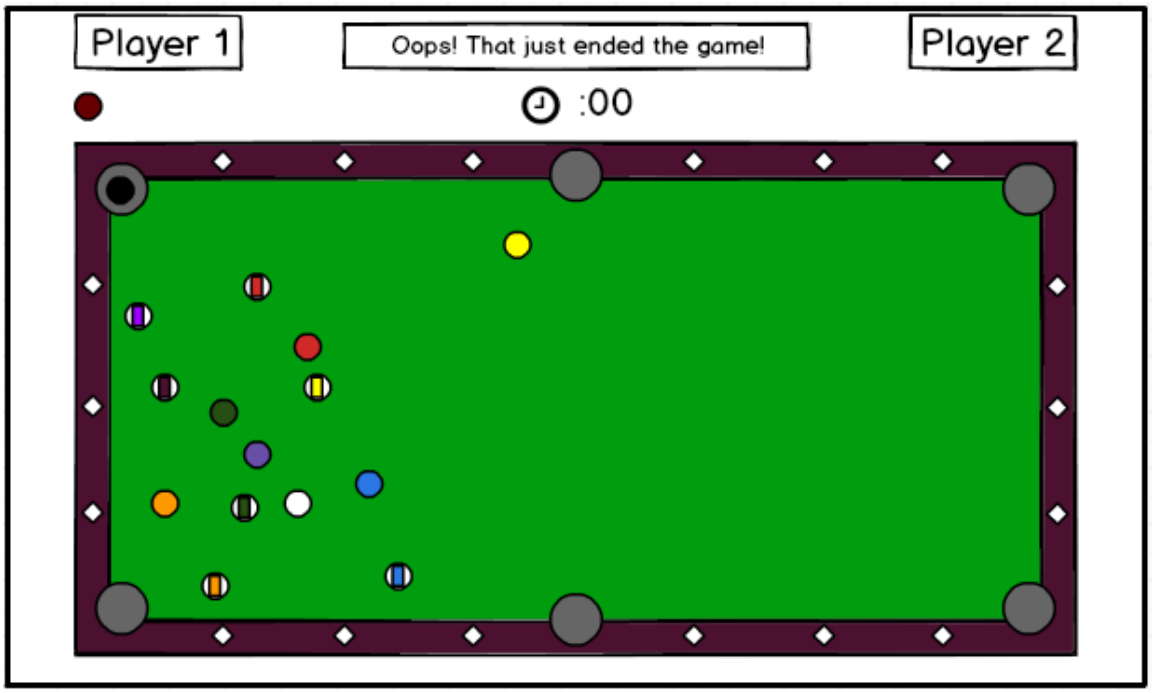
Player 1 misses, so it is now Player 2's turn. Player 2's goal is to pocket all the striped balls and then the 8-ball.



Player 2 gets distracted and runs out of time! Her turn is over.

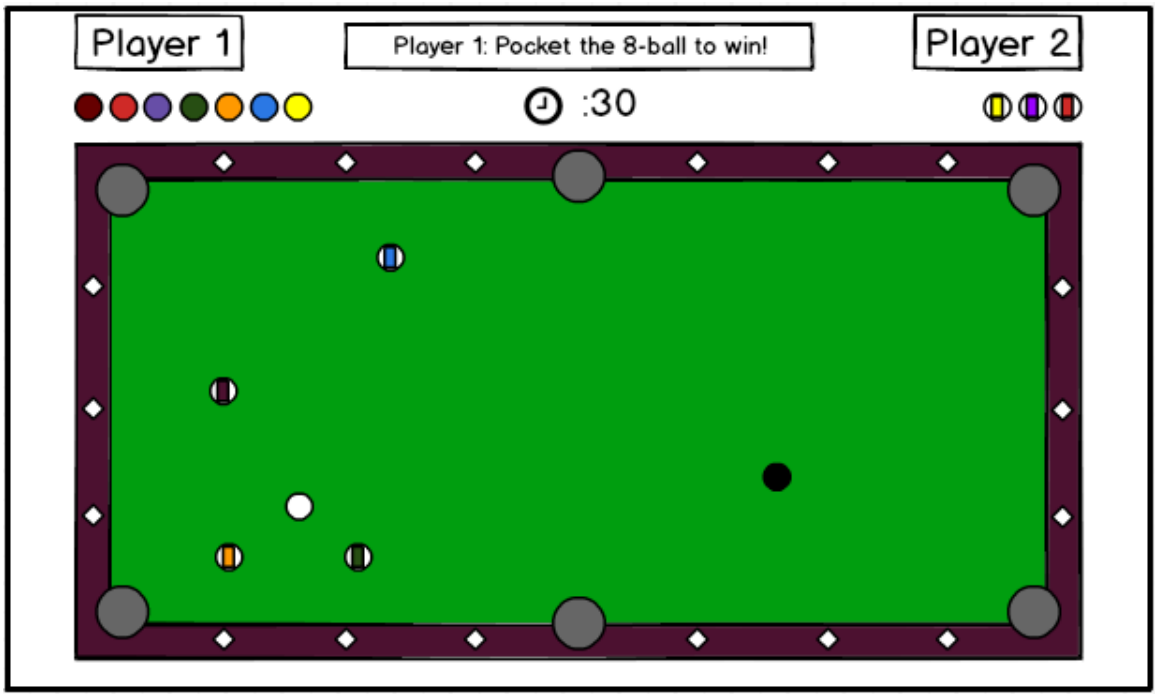


This time, Player 1 makes a mistake of his own by pocketing the cue ball. Because of this, Player 2 places the ball wherever she chooses inside the designated zone.



Uh oh! Player 2 pockets the black 8-ball before pocketing all the other striped balls.

The game is over - Player 2 loses, and Player 1 wins!
Alternate ending: Player 2 does not prematurely pocket the 8-ball, and the game continues. Soon, Player 1 pockets all of her solid balls and only has the 8-ball remaining.



Player 1 must pocket the 8-ball to defeat Player 2.

Player 1 successfully pockets the 8-ball and wins the game.

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
  - 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
    - final double radius: set to 1 on initialization
    - 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
- class PlayerTurn
  - Instance variables:
    - GameBoard gameBoard
  - constructor: PlayerTurn(CueStick cueStick, GameBoard gameBoard) - transfers the velocity/direction of the cue stick to the cue ball.
  - Methods
    - checkCollisions(): loops through all ball/boundary objects and uses the Collisions class to check whether any collisions have occurred and how the momentums of the objects should be updated.
    - updateAllBallPositions(): updates positions of the balls on the table depending on their directions and velocities
    - makeTurn(): loops through checkCollisions() and updateAllBallPositions() every millisecond until all objects are at rest. Checks the final positions of the cue ball and numbered balls. Updates player scores. Update next player to move.
- class GameBoard
  - Instance Variables:
    - Ball[] balls: 16 balls
    - Boundary[] boundaries: size 4
    - Pocket[] pockets: size 6
  - constructor: GameBoard()
- class Boundary
  - Instance Variables:
    - Point[] 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 Player
    - Instance variables:
        - int playerBallType: 1 if striped, 0 if solid, null otherwise (if not set yet)
        - ArrayList<Ball> pocketedBalls
        - canPocketEightBall
    - constructor: Player()
- class Collisions
    - Instance variables:
        - Object[] objects
    - Constructor: Collision(Object[] objects)
    - 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
        - playGame(): the main game loop that will run, gets information from the controllers to update the view, calls makeTurn() until game ends
        - makeTurn(): start timer, place CueBall if scratched, set CueBallStick instance variables, stop timer, create a PlayerTurn object
        - pauseGame(): pauses entire game
        - endGame(): ends the entire game

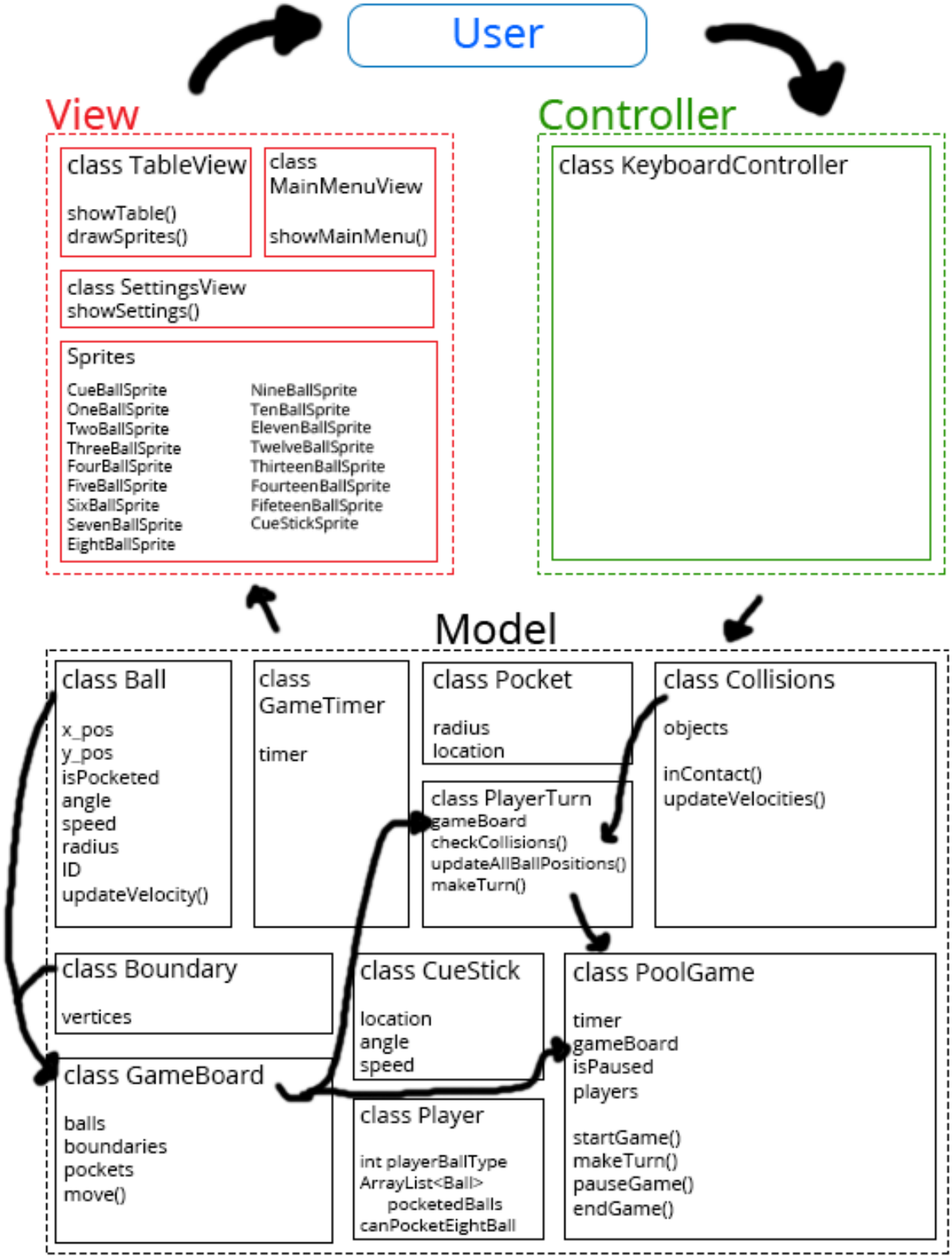
VIEW

- CueBallSprite
- OneBallSprite
- TwoBallSprite
- ThreeBallSprite
- FourBallSprite
- FiveBallSprite
- SixBallSprite
- SevenBallSprite
- EightBallSprite
- NineBallSprite
- TenBallSprite

- ElevenBallSprite
- TwelveBallSprite
- ThirteenBallSprite
- FourteenBallSprite
- FifteenBallSprite
- CueStickSprite
- TableBoardView
    - showTable(): shows the table
    - drawSprites(): constantly checks on individual sprites and updates the view to reflect their positions
- MainMenuView
    - showMainMenu(): shows the main menu
- SettingsView
    - showSettings(): shows the settings

CONTROLLER

- MouseController


User
View
class TableView
showTable()
drawSprites()
class MainMenuView
showMainMenu()
class SettingsView
showSettings()
Sprites
CueBallSprite
OneBallSprite
TwoBallSprite
ThreeBallSprite
FourBallSprite
FiveBallSprite
SixBallSprite
SevenBallSprite
EightBallSprite
NineBallSprite
TenBallSprite
ElevenBallSprite
TwelveBallSprite
ThirteenBallSprite
FourteenBallSprite
FifeteenBallSprite
CueStickSprite
Controller
class KeyboardController
Model
class Ball
x_pos
y_pos
isPocketed
angle
speed
radius
ID
updateVelocity()
class GameTimer
timer
class Pocket
radius
location
class PlayerTurn
gameBoard
checkCollisions()
updateAllBallPositions()
makeTurn()
class Collisions
objects
inContact()
updateVelocities()
class Boundary
vertices
class GameBoard
balls
boundaries
pockets
move()
class CueStick
location
angle
speed
class Player
int playerBallType
ArrayList<Ball>
pocketedBalls
canPocketEightBall
class PoolGame
timer
gameBoard
isPaused
players
startGame()
makeTurn()
pauseGame()
endGame()

Tests

Ball Class
- constructor: Ball(int id)
  - should assign inputted ID to the ball
  - should set default radius of 1
  - 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

Player Class
- constructor: Player()
  - should assign default values to the player

Collisions Class
- constructor: Collision(Object[] objects)
  - should assign objects to the collision
  - should throw when not given Ball or Boundary objects
- Constructor: Collision(Object[] objects)
  - 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();
      b1.setPosition(50, 50);
      Ball b2 = new Ball();
      b2.setPosition(49, 50); //radius of 1
      Object[] objects = {b1, b2};
      Collision c = new Collision(objects);
      boolean expected = true;

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

- TwoObjectsWithDifferentPositionsDoNotCollideAndShouldReturnFalse()
  Ball b1 = new Ball();
  b1.setPosition(50, 50);
  Ball b2 = new Ball();
  b2.setPosition(40,50);
  Object[] objects = {b1, b2};
  Collision c = new Collision(objects);
  boolean expected = false;
  AssertEquals("Two objects with different positions collided", expected,
  c.inContact();)
- ThreeObjectsWithNotSamePositionsDoNotCollideAndShouldReturnFalse()
  Ball b1 = new Ball();
  b1.setPosition(50, 50);
  Ball b2 = new Ball();
  b2.setPosition(40,50);
  Ball b3 = new Ball();
  b3.setPosition(40,50);
  Object[] objects = {b1, b2, b3}
  Collision c = new Collision(objects);
  boolean expected = false;
  AssertEquals("Three objects not in same positions collided", expected,
  c.inContact();
- CollisionBetweenBallAndBoundaryShouldReturnTrue()
  Ball ball = new Ball();
  ball.setPosition(49, 50);
  Boundary bound = new Boundary(new Point(50, 0), new Point(50, 100);
  Object objects = {ball, bound};
  Collision c = new Collision(objects);
  boolean expected = true;
  AssertEquals("Boundary and ball collision did not return true",
  expected,c.inContact());

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