```
//
//  Grid.m
//  ninabaculinao
//
//  Created by Nina Baculinao on 7/10/14.
//  Copyright (c) 2014 Apportable. All rights reserved.
//

#import "Grid.h"
#import "Dice.h"

@implementation Grid {

    CGFloat _tileWidth; //37
    CGFloat _tileHeight; //37
    CGFloat _tileMarginVertical; //0.9285714285714286
    CGFloat _tileMarginHorizontal; //0.6153846153846154 2

    NSMutableArray *_gridArray; // a 2d array
    NSNull *_noTile;

    float timer;
    float timeSinceDrop;
    float dropInterval;

    Dice *_currentDie1;
    Dice *_currentDie2;
    NSMutableArray *_dicePair;

    CGPoint oldTouchPosition;
}

// two constants to describe the amount of rows and columns
static const NSInteger GRID_ROWS = 12;
static const NSInteger GRID_COLUMNS = 6;

- (void)didLoadFromCCB{

    timer = 0;
    timeSinceDrop = -0.2;
    dropInterval = 0.5;

    _dicePair = [NSMutableArray array];

    self.userInteractionEnabled = TRUE;

    [self setupGrid];

    // Fill array with null tiles
    _noTile = [NSNull null];
    _gridArray = [NSMutableArray array];

    for (NSInteger i = 0; i < GRID_ROWS; i++) {
        _gridArray[i] = [NSMutableArray array];
        for (NSInteger j = 0; j < GRID_COLUMNS; j++) {
            _gridArray[i][j] = _noTile;
```

```
        }
    }

    [self spawnDice];

    // debugging indexvalidandunoccupied method
//  for (NSInteger i = 0; i <= GRID_ROWS; i++) {
//      for (NSInteger j = 0; j <= GRID_COLUMNS; j++) {
//            BOOL free = [self indexValidAndUnoccupiedForRow:i andColumn:j];
//            CCLOG(@"Row %ld and column %ld free? %d", (long)i, (long)j, free);
//        }
//}

//    // listen for swipes to the left
//    UISwipeGestureRecognizer * swipeLeft= [[UISwipeGestureRecognizer
    alloc]initWithTarget:self action:@selector(swipeLeft)];
//    swipeLeft.direction = UISwipeGestureRecognizerDirectionLeft;
//    [[[CCDirector sharedDirector] view] addGestureRecognizer:swipeLeft];
//    // listen for swipes to the right
//    UISwipeGestureRecognizer * swipeRight= [[UISwipeGestureRecognizer
    alloc]initWithTarget:self action:@selector(swipeRight)];
//    swipeRight.direction = UISwipeGestureRecognizerDirectionRight;
//    [[[CCDirector sharedDirector] view] addGestureRecognizer:swipeRight];
//    // listen for swipes down
//    UISwipeGestureRecognizer * swipeDown= [[UISwipeGestureRecognizer
    alloc]initWithTarget:self action:@selector(swipeDown)];
//    swipeDown.direction = UISwipeGestureRecognizerDirectionDown;
//    [[[CCDirector sharedDirector] view] addGestureRecognizer:swipeDown];
////    // listen for swipes up
//    UISwipeGestureRecognizer * swipeUp= [[UISwipeGestureRecognizer
    alloc]initWithTarget:self action:@selector(swipeUp)];
//    swipeUp.direction = UISwipeGestureRecognizerDirectionUp;
//    [[[CCDirector sharedDirector] view] addGestureRecognizer:swipeUp];

}

# pragma mark - Update method

- (void) update:(CCTime) delta {
    timer += delta;
    timeSinceDrop += delta;

    if (timeSinceDrop > dropInterval) {
        [self dieFallDown];
        timeSinceDrop = 0;
        if (![self canBottomMove]) {
            [self spawnDice];
            timeSinceDrop = -0.2;
            dropInterval = 1.0;
        }

    }
}

# pragma mark - Create grid
```

```
- (void)setupGrid
{
    _tileWidth = 37.f;
    _tileHeight = 37.f;

    // calculate the margin by subtracting the block sizes from the grid size
    _tileMarginHorizontal = (self.contentSize.width - (GRID_COLUMNS * _tileWidth)
        ) / (GRID_COLUMNS+1);
    _tileMarginVertical = (self.contentSize.height - (GRID_ROWS * _tileHeight)) /
        (GRID_ROWS+1);

    // set up initial x and y positions
    float x = _tileMarginHorizontal;

    float y = _tileMarginVertical;

    // initialize the array as a blank NSMutableArray
    _gridArray = [NSMutableArray array];

    for (NSInteger i = 0; i < GRID_ROWS; i++) {
        // iterate through each row
        // create 2d array by putting array into array
        _gridArray[i] = [NSMutableArray array];
        x = _tileMarginHorizontal;

        for (NSInteger j = 0; j < GRID_COLUMNS; j++) {
            //  iterate through each column in the current row
            x+= _tileWidth + _tileMarginHorizontal; // after positioning a block
                increase x variable
        }

        y+= _tileHeight + _tileMarginVertical; // after completing row increase y
            variable
    }
}

# pragma mark - Create random Dice

-(Dice*) randomizeNumbers {
    NSInteger random = arc4random_uniform(6)+1;
    Dice *die;
    switch(random)
    {
        case 1:
            die = (Dice*) [CCBReader load:@"Dice/One"];
            CCLOG(@"Face: 1");
            break;
        case 2:
            die = (Dice*) [CCBReader load:@"Dice/Two"];
            CCLOG(@"Face: 2");
            break;
        case 3:
            die = (Dice*) [CCBReader load:@"Dice/Three"];
            CCLOG(@"Face: 3");
            break;
        case 4:
```

```
            die = (Dice*) [CCBReader load:@"Dice/Four"];
            CCLOG(@"Face: 4");
            break;
        case 5:
            die = (Dice*) [CCBReader load:@"Dice/Five"];
            CCLOG(@"Face: 5");
            break;
        case 6:
            die = (Dice*) [CCBReader load:@"Dice/Six"];
            CCLOG(@"Face: 6");
            break;
        default:
            die = (Dice*) [CCBReader load:@"Dice/Dice"];
            CCLOG(@"WHY IS IT AT DEFAULT");
            break;
    }
    return die;
}

- (Dice*) addDieAtTile:(NSInteger)column row:(NSInteger)row {
    Dice* die = [self randomizeNumbers];
    _gridArray[row][column] = die;
    die.row = row;
    die.column = column;
    die.scale = 0.f;
    [self addChild:die];
    die.position = [self positionForTile:column row:row];
    CCActionDelay *delay = [CCActionDelay actionWithDuration:0.3f];
    CCActionScaleTo *scaleUp = [CCActionScaleTo actionWithDuration:0.2f scale:1.f
        ];
    CCActionSequence *sequence = [CCActionSequence actionWithArray:@[delay,
        scaleUp]];
    [die runAction:sequence];
    return die;

//    CCActionMoveTo *fall = [CCActionMoveTo actionWithDuration:5.0f
    position:ccp(die.position.x, 0)];
//    CCActionMoveTo *fallSequence = [CCActionSequence actionWithArray:@[delay,
    fall]];
//    [die runAction:fallSequence];
}

//
//- (void)fallingDie {
//    BOOL falling = FALSE;
//    while (!falling) {
//
//    }
//}

- (void)spawnDice {
    BOOL spawned = FALSE;
    while (!spawned) {
        NSInteger firstRow = 11;
        NSInteger firstColumn = arc4random_uniform(5); // int bt 0 and 4
        CCLOG(@"First Column %ld, Row %ld", (long)firstColumn, (long)firstRow);
```

```
            NSInteger nextRow = firstRow - arc4random_uniform(2);
            NSInteger nextColumn;
            if (firstRow != nextRow) { // has to be vertical
                nextColumn = firstColumn;
            } else { // has to be horizontal
                nextColumn = firstColumn+1;
            }
            CCLOG(@"Next Column %ld, Row %ld", (long)nextColumn, (long)nextRow);

            BOOL positionFree = (_gridArray[firstRow][firstColumn] == _noTile);
            BOOL nextPositionFree = (_gridArray[nextRow][nextColumn] == _noTile);
            if (positionFree && nextPositionFree) {
                _currentDie1 = [self addDieAtTile:firstColumn row:firstRow];
                _currentDie2 = [self addDieAtTile:nextColumn row:nextRow];
                _dicePair = [NSMutableArray arrayWithObjects:_currentDie1,
                    _currentDie2, nil];
                spawned = TRUE;
            } else {
                CCLOG(@"Game Over");
                break;
            }

        }
    }

    # pragma mark - Position for tile from (column, row) to ccp(x,y)

    - (CGPoint)positionForTile:(NSInteger)column row:(NSInteger)row {
        float x = _tileMarginHorizontal + column * (_tileMarginHorizontal +
            _tileWidth);
        float y = _tileMarginVertical + row * (_tileMarginVertical + _tileHeight);
        return CGPointMake(x,y);
    }

    # pragma mark - Falling dice

    - (void) dieFallDown {
        BOOL bottomCanMove = [self canBottomMove];
        if (bottomCanMove) {
            _gridArray[_currentDie1.row][_currentDie1.column] = _noTile;
            _gridArray[_currentDie2.row][_currentDie2.column] = _noTile;

            _currentDie1.row--;
            _gridArray[_currentDie1.row][_currentDie1.column] = _currentDie1;
            _currentDie1.position = [self positionForTile:_currentDie1.column row:
                _currentDie1.row];

            _currentDie2.row--;
            _gridArray[_currentDie2.row][_currentDie2.column] = _currentDie2;
            _currentDie2.position = [self positionForTile:_currentDie2.column row:
                _currentDie2.row];
        }

        //    for (Dice *die in _dicePair) {
        //        if (bottomCanMove) {
        //            _gridArray[die.row][die.column] = _noTile;
```

```
    //            die.row--;
    //            _gridArray[die.row][die.column] = die;
    //            die.position = [self positionForTile:die.column row:die.row];
    //        }
    //
    //    }
}

- (BOOL) canBottomMove {
    if (_currentDie1.row != _currentDie2.row) {
        if (_currentDie1.row > _currentDie2.row) {
            return [self indexValidAndUnoccupiedForRow:_currentDie2.row-1
                andColumn:_currentDie2.column];
        } else {

            return [self indexValidAndUnoccupiedForRow:_currentDie1.row-1
                andColumn:_currentDie1.column];
        }
    }
    else {
        return [self indexValidAndUnoccupiedForRow:_currentDie2.row-1 andColumn:
            _currentDie2.column] && [self indexValidAndUnoccupiedForRow:
            _currentDie1.row-1 andColumn:_currentDie1.column];
    }
}

//- (void) dieFalling: Dice*(die) fromColumn:(NSInteger)column andRow:
    (NSInteger)row {
//    for (NSInteger i = row; i >= 0; i++) {
//        _gridArray[column][row-i] = _gridArray[column][row];
//        _gridArray[column][row] = _noTile;
//        CGPoint newPosition = [self positionForTile:column row:row];
//        CCActionMoveTo *moveTo = [CCActionMoveTo actionWithDuration:0.2f
    position:newPosition];
//        [die runAction:moveTo];
//    }
//}

# pragma mark - Touch and swipe handling

- (void)touchBegan:(UITouch *)touch withEvent:(UIEvent *)event {
    oldTouchPosition = [touch locationInNode:self];
}

- (void)touchEnded:(UITouch *)touch withEvent:(UIEvent *)event {
    CGPoint newTouchPosition = [touch locationInNode:self];
    float xdifference = oldTouchPosition.x - newTouchPosition.x;
    float ydifference = oldTouchPosition.y - newTouchPosition.y;
    NSInteger column = ((newTouchPosition.x - _tileMarginHorizontal) /
        (_tileWidth + _tileMarginHorizontal));
    if (column > 5) {
        column = 5;
    } else if (column < 0) {
        column = 0;
    }
```

```
    if (ydifference > 0.2*(self.contentSize.height) || ydifference < -0.2*(self.
        contentSize.height)) {
        [self dropDown];
    } else if (xdifference > 0.3*(self.contentSize.width)) {
        [self swipeLeftTo:column];
    } else if (xdifference > 0.1*(self.contentSize.width) && xdifference < 0.3*
        (self.contentSize.width)) {
        [self swipeLeft];
    } else if (xdifference < -0.3*(self.contentSize.width)) {
        [self swipeRightTo:column];
    } else if (xdifference < -0.1*(self.contentSize.width) && xdifference > -0.3*
        (self.contentSize.width)){
        [self swipeRight];
    } else {
        [self rotate];
    }
}

- (void)swipeLeftTo:(NSInteger)column {
    while (_currentDie1.column > column && _currentDie2.column > column) {
        [self swipeLeft];
    }
}


- (void)swipeLeft {
//    [self move:ccp(-1, 0)];
    BOOL bottomCanMove = [self canBottomMove];
    BOOL canMoveLeft = [self indexValidAndUnoccupiedForRow:_currentDie2.row
        andColumn:_currentDie2.column-1] && [self indexValidAndUnoccupiedForRow:
        _currentDie1.row andColumn:_currentDie1.column-1];
    if (bottomCanMove && canMoveLeft) {
        _gridArray[_currentDie1.row][_currentDie1.column] = _noTile;
        _gridArray[_currentDie2.row][_currentDie2.column] = _noTile;

        _currentDie1.column--;
        _gridArray[_currentDie1.row][_currentDie1.column] = _currentDie1;
        _currentDie1.position = [self positionForTile:_currentDie1.column row:
            _currentDie1.row];

        _currentDie2.column--;
        _gridArray[_currentDie2.row][_currentDie2.column] = _currentDie2;
        _currentDie2.position = [self positionForTile:_currentDie2.column row:
            _currentDie2.row];
    }
}

- (void)swipeRightTo:(NSInteger)column {
    while (_currentDie1.column < column && _currentDie2.column < column) {
        [self swipeRight];
    }
}


- (void)swipeRight {
//    [self move:ccp(1, 0)];
```

```
    BOOL bottomCanMove = [self canBottomMove];
    BOOL canMoveRight = [self indexValidAndUnoccupiedForRow:_currentDie2.row
        andColumn:_currentDie2.column+1] && [self indexValidAndUnoccupiedForRow:
        _currentDie1.row andColumn:_currentDie1.column+1];
    if (bottomCanMove && canMoveRight) {
        _gridArray[_currentDie1.row][_currentDie1.column] = _noTile;
        _gridArray[_currentDie2.row][_currentDie2.column] = _noTile;

        _currentDie1.column++;
        _gridArray[_currentDie1.row][_currentDie1.column] = _currentDie1;
        _currentDie1.position = [self positionForTile:_currentDie1.column row:
            _currentDie1.row];

        _currentDie2.column++;
        _gridArray[_currentDie2.row][_currentDie2.column] = _currentDie2;
        _currentDie2.position = [self positionForTile:_currentDie2.column row:
            _currentDie2.row];
    }
}

- (void)dropDown {
//    [self move:ccp(0, -1)];
    dropInterval= 0.001;
}


//- (void)swipeUp {
//    [self move:ccp(1,0)];
//}

- (void)rotate {
    BOOL bottomCanMove = [self canBottomMove];
    BOOL isRotating = true;
    if (isRotating) {
        [self unschedule:@selector(dieFallDown)];
    } else {
        [self schedule:@selector(dieFallDown) interval:0.5f];
    }

    if (isRotating && bottomCanMove) {
        if (_currentDie2.column > _currentDie1.column) {
            // [1][2] --> [1]
            //            [2]
            _gridArray[_currentDie2.row][_currentDie2.column] = _noTile;
            _currentDie2.row--; _currentDie2.column--;
            _gridArray[_currentDie2.row][_currentDie2.column] = _currentDie2;
            _currentDie2.position = [self positionForTile:_currentDie2.column
                row:_currentDie2.row];
        } else if (_currentDie1.row > _currentDie2.row) {
            // [1]
            // [2] --> [2][1]
            if (_currentDie2.column == 0) {
                _gridArray[_currentDie1.row][_currentDie1.column] = _noTile;
                _currentDie1.row--; _currentDie1.column++;
                _gridArray[_currentDie1.row][_currentDie1.column] = _currentDie1;
                _currentDie1.position = [self positionForTile:_currentDie1.column
```

```
                    row:_currentDie1.row];
            } else if (_currentDie1.row == 11) {
                _gridArray[_currentDie1.row][_currentDie1.column] = _noTile;
                _currentDie1.row--; _currentDie1.column--;
                _gridArray[_currentDie1.row][_currentDie1.column] = _currentDie1;
                _currentDie1.position = [self positionForTile:_currentDie1.column
                    row:_currentDie1.row];
            }
            else {
                _gridArray[_currentDie2.row][_currentDie2.column] = _noTile;
                _currentDie2.row++; _currentDie2.column--;
                _gridArray[_currentDie2.row][_currentDie2.column] = _currentDie2;
                _currentDie2.position = [self positionForTile:_currentDie2.column
                    row:_currentDie2.row];
            }
        } else if (_currentDie1.column > _currentDie2.column) {
            // [2][1] --> [2]
            //            [1]
            _gridArray[_currentDie2.row][_currentDie2.column] = _noTile;
            _currentDie2.row++; _currentDie2.column++;
            _gridArray[_currentDie2.row][_currentDie2.column] = _currentDie2;
            _currentDie2.position = [self positionForTile:_currentDie2.column
                row:_currentDie2.row];
        } else {
            // [2]
            // [1]  --> [1][2] means die2 moves
            if (_currentDie2.column == 5) {
                _gridArray[_currentDie1.row][_currentDie1.column] = _noTile;
                _currentDie1.row++; _currentDie1.column--;
                _gridArray[_currentDie1.row][_currentDie1.column] = _currentDie1;
                _currentDie1.position = [self positionForTile:_currentDie1.column
                    row:_currentDie1.row];
            } else {
                _gridArray[_currentDie2.row][_currentDie2.column] = _noTile;
                _currentDie2.row--; _currentDie2.column++;
                _gridArray[_currentDie2.row][_currentDie2.column] = _currentDie2;
                _currentDie2.position = [self positionForTile:_currentDie2.column
                    row:_currentDie2.row];
            }
        }
        isRotating = false;
    }
}


# pragma mark - Move dice

//- (void)move:(CGPoint)direction {
//    // apply negative vector until reaching boundary, this way we get the tile
    that is the furthest away
//    // bottom left corner
//    NSInteger currentX = 0;
//    NSInteger currentY = 0;
//    // Move to relevant edge by applying direction until reaching border
//    while ([self indexValid:currentX y:currentY]) {
//        CGFloat newX = currentX + direction.x;
```

```
//        CGFloat newY = currentY + direction.y;
//        if ([self indexValid:newX y:newY]) {
//            currentX = newX;
//            currentY = newY;
//        } else {
//            break;
//        }
//    }
//    // store initial row value to reset after completing each column
//    NSInteger initialY = currentY;
//    // define changing of x and y value (moving left, up, down or right?)
//    NSInteger xChange = -direction.x;
//    NSInteger yChange = -direction.y;
//    if (xChange == 0) {
//        xChange = 1;
//    }
//    if (yChange == 0) {
//        yChange = 1;
//    }
//    // visit column for column
//    while ([self indexValid:currentX y:currentY]) {
//        while ([self indexValid:currentX y:currentY]) {
//            // get tile at current index
//            Dice *die = _gridArray[currentX][currentY];
//            if ([die isEqual:_noTile]) {
//                // if there is no tile at this index -> skip
//                currentY += yChange;
//                continue;
//            }
//            // store index in temp variables to change them and store new
    location of this tile
//            NSInteger newX = currentX;
//            NSInteger newY = currentY;
//            /* find the farthest position by iterating in direction of the
    vector until we reach border of grid or an occupied cell*/
//            while ([self indexValidAndUnoccupied:newX+direction.x y:newY
    +direction.y]) {
//                newX += direction.x;
//                newY += direction.y;
//            }
//            if (newX != currentX || newY !=currentY) {
//                [self moveDice:die fromIndex:currentX oldY:currentY newX:newX
    newY:newY];
//            }
//            // move further in this column
//            currentY += yChange;
//        }
//        // move to the next column, start at the inital row
//        currentX += xChange;
//        currentY = initialY;
//    }
//}
//
//- (void)moveDice:(Dice *)die fromIndex:(NSInteger)oldX oldY:(NSInteger)oldY
    newX:(NSInteger)newX newY:(NSInteger)newY {
//    _gridArray[newX][newY] = _gridArray[oldX][oldY];
```

```
//    _gridArray[oldX][oldY] = _noTile;
//    CGPoint newPosition = [self positionForTile:newX row:newY];
//    CCActionMoveTo *moveTo = [CCActionMoveTo actionWithDuration:2.0f
    position:newPosition];
//    [die runAction:moveTo];
//}

# pragma mark - Check indexes

- (BOOL)indexValidForRow:(NSInteger)row AndColumn:(NSInteger)column {
    BOOL indexValid = YES;
    if(row < 0 || column < 0 || row >= GRID_ROWS || column >= GRID_COLUMNS)
    {
        indexValid = NO;
    }
    return indexValid;
}
//    BOOL indexValid = TRUE;
//    indexValid &= x >= 0;
//    indexValid &= y >= 0;
//    if (indexValid) {
//        indexValid &= x < GRID_ROWS;
//        if (indexValid) {
//            indexValid &= y < GRID_COLUMNS;
//        }
//    }


- (BOOL)indexValidAndUnoccupiedForRow:(NSInteger)row andColumn:(NSInteger)column
    {
    BOOL indexValid = [self indexValidForRow:row AndColumn:column];
    if (!indexValid) {
        return FALSE;
    }
    else {
        BOOL unoccupied = [_gridArray[row][column] isEqual:_noTile] ||
            [_gridArray[row][column] isEqual:_currentDie2]  || [_gridArray[row]
            [column] isEqual:_currentDie1] ;
        return unoccupied;

    }
}

@end
```