# SYSTEM ARCHITECTURE

## 1.1 Functional Properties

**User Accounts:** The Google Play Game Services API, provides the user with the capability to sign in using their google account, and from there the two users can be connected and play the game. The API also provides a mechanism for invitations to initiate gameplay.

**Paddle:** The board owns references to two paddles on for the player and one for the opponent, and depending on the ball’s position, if it matches with either the user’s or opponent's paddle positions then it tells the corresponding paddle to respond to a potential hit.

**Prisons:** A Prison base class will be used to represent the different properties of a prison. Real and fake prisons will be two classes that inherit from the Prison base class. The fake and real prisons will be displayed in the same fashion but will handle hit events differently.

**Ball:** The board owns a reference to the ball, and when the ball’s position matches with the position of either the paddle, bricks or prisons the corresponding object gets notified and hit detection is implemented.

**Board:** The board is implemented as a fixed number of rows and columns that divide the screen into squares. Each square is a GridItem. Different items such as bricks then inherit from the GridItem class.

**Bricks:**Bricks inherit from the GridItem base class. Due to the fact that there are two types of blocks which are the pure blocking blocks and the power releasing blocks, there is a need for a brick base class that these two types of bricks inherit from. The two types of bricks are displayed in a similar fashion but implement hit behaviour differently.

**Powerups:** All power-ups inherit from the same powerup base class, but implement different behaviour when they are acquired.

**Home Screen:** The home screen is a linear layout that provides the users with the single player and multiplayer modes of play.

## 1.2 Non-Functional Properties

**Performance:** In order to make sure the application is responsive, the Google Play Games Services API was used to take care of the underlying connections between users. The API provides real-time multiplayer services by creating a game room that users can join. The room sets up a peer-to-peer mesh network where users can directly communicate with each other. Other performance enhancing decisions that were made include the avoidance of any object instantiation in the onDraw() method as it is called frequently.

**Scalability:** Using the Google Play Games Services API makes sure that the application is scalable. This is because this API already has proven scalability. There are many games on Google Games with millions of users who can enjoy a seamless and smooth multiplayer experience.

**Efficiency:** In terms of efficiency, it is true that using the Google Games Services can sometimes cause excessive battery drainage, but solutions do exist to remedy the problem. Firstly, the user is able to clear the Google Play Services cache that can become bloated with excess unnecessary information. Another option available to users is turning off Auto-sync for various account such as email, calendar and google drive.

#### **Heterogeneity:** To make sure that the game is able to run consistently on different devices, agrid system with a fixed number of rows and columns was used. The board is vertically centered so that regardless of which screen it is on, each grid item is square.

## 1.3 Component Diagram

# 2.0 DESIGN

## 2.1 Ball

The ball class is one of the major functional component implementations in our system. The code in this class handles it’s drawing, animation and logic to change animation patterns.

### 2.1.1 Structure

* Drawing   
      Ball extends Android’s View class so that it can implement it’s own *OnDraw()* method to allow for custom drawing.   
      A RectF structure, initialized in the constructor, is used to define a rectangle that is drawn as a circle in the *OnDraw()* method. This RectF’s coordinates are passed as parameters to the Ball’s constructor.  
      The color of the Ball is handled by the Paint object. Currently, it is hardcoded in the Ball’s constructor but we plan to pass it as a parameter to allow for different colored Balls.
* Speed  
      This property is used to calculate animation times for the animators attached to the Ball. By increasing the Speed property, we decrease the animation time for the Animators, thereby ‘increasing’ the speed at which the ball is moving around the screen.
* Animators  
      Android’s ValueAnimator objects are added to the Ball (since it extends a View). The reason we use ValueAnimators instead of the ObjectAnimators is that it updates the coordinates of the Ball (a View) when it runs the animation.  
      There are two different animators for animating the Ball’s X and Y directions. They are both attached in *addAnimators()* method. For each of the Animators, we add Update Listeners which update the position properties of the Ball. These position properties are used when drawing the Ball. So a change of position and a call to the *OnDraw()* method will effectively make the Ball move on the screen.
* GetEndX(), GetEndY()  
      Since the Animators define an animation by its start and end coordinates, we need methods to calculate those. The starting coordinates of an animation can just be the current location of the Ball, we can get those by its position properties. The *GetEndX()* and *GetEndY()* methods the direction the Ball is moving in (up or down, right or left) and calculate the coordinates at which the Animator should stop animating the Ball ie. the end point
* Reverse Directions  
      Since there can only be two directions on the X and Y plane, we use 1 and -1 for each to signify whether a ball is moving right or left and up or down. These directions are stored in the *dir[2]*  array. The *reverseX()* and *reverseY()* methods just switch the value of *dir[0]* and *dir[1]* from 1 to -1 and vice versa.

## 2.2 Board

The Board class is a major functional component that models the game state and the layout of the GridItems in the MainActivity. A cuustom grid coordinate system is used so that the game would look consistent on different sized phone screens when drawn with BoardView, and collision detection is simplified. It also handles the collision interactions between the Ball and GridItem classes.

### 2.2.1 Structure

#### ArrayList<ArrayList<GridItem>> grid

A 2D array that holds GridItems (blanks, Bricks, and Jails). Its dimensions are defined in the init function. This array is used to help the Board locate and notify GridItems that are potentially hit by the Ball.

#### void init(Context)

Obtains the dimensions of the screen in pixels. It uses this information to calculate appropriate grid sizes that would create a 16:9 board that has square shaped GridItems. It also calculates the vertical offset required to vertically center the board on phone screens that do not follow the 16:9 screen ratio.

#### void initBoard(Context, FrameLayout)

Currently uses a BrickFactory to randomly create 10 SquareBricks on each player’s section of the board and adds them to the provided FrameLayout.

#### Board.Boundaries getBoardBoundaries()

Returns a Boundaries object which has the y-axis pixel location of the BoardTop, OpponentTop, PlayerTop, and BoardBottom. This is needed for the Ball to know where the top and bottom of the board are, and for the Paddle to know along which y-value (OpponentTop and PlayerTop) the Paddle should move horizontally on.

#### Boolean isHit(float, float, float)

Given the Ball’s position and size in pixels, isHit calculates which GridItems are potentially hit by the ball and notifies them that they may have been hit. If they have been hit, the function will return True, and the Ball will reverse its direction.

#### float getGridItemSize()

Returns the GridItem size as calculated during the initialization of the Board object. Since the GridItem is square, its width and height are the same.

#### float getNumColumns(), float getNumRows()

Returns the number of columns and rows (respectively) on the Board.

### 2.2.2 Design & Rationale

The Board is a singleton because there should only be one game state. This prevents duplication. All of its parameters are private and only retrievable via getters, which makes it unmodifiable by other classes. Other classes access the getters via the Helper class, so that they do not have to have a copy of the Board object.

A fixed number of columns and rows have been carefully chosen so that each GridItem is a decently sized square and the Board follows a 16:9 ratio, which allows for a full-screen experience for most phones. In the case that the phone doesn’t have a 16:9 aspect ratio, vertical gaps will be added to ensure that the grid items remain square and the board is vertically centered.

Also, with our custom grid coordinate system, we can easily translate a Ball’s pixel position to a grid coordinate, and notify the GridItem in that grid coordinate that it may have been hit. In the worst case, only 4 GridItems will be notified of a potential hit. When communicating between phones of different screen sizes about the positions of objects, the grid coordinate makes it easy to calculate where the object should be drawn on their own screens.

The Board sections/background are drawn by the BoardView’s only function, onDraw. The reason for separating the BoardView from the Board is because view objects can be destroyed on the end of an activity. To ensure the state of the game and board can persist across different activities via Intent objects, we separated the model and the view of the board into those two classes.

To allow users to pause the game or to build the board, an enum will be added to the Board class to denote the state of the game, which will let the MainActivity change its behavior.

## 2.3 GridItems

The GridItems class is the parent of Jail (will be implemented) and Brick classes. Polymorphism allows the Board class to treat all GridItems the same, while each GridItem may handle the events differently. This decreases logical cohesion. A GridItem object that is neither a Jail nor a Brick is a blank spot, therefore, its onHit function is an empty one.

### 2.3.1 Structure

#### int row, int column

The grid coordinate position of the GridItem.

#### boolean onHit(ArrayList<int[]>)

An empty function, so that children are enforced to implement this function, and so that the Board can treat all GridItems the same. It accepts an array list of the pixel positions of the Ball’s bounding box’s 4 corners.

#### int[] getPosition()

Returns the x and y position of the GridItem, according to our custom grid coordinate system.

## 2.4 Brick (and its children classes)

The Brick class is an abstract one that has five children classes: Jail, Square, LeftUpperTriangle, LeftLowerTriangle, RightUpperTriangle, and RightLowerTriangle. The separation was required since they all handle onHit and onDraw differently. Brick objects inherit all parameters from GridItems.

### 2.4.1 Structure

#### int hp, int opacity

Depending on the number of times the brick has been hit, opacity will be zero so that it doesn’t show up on the board anymore. When the hp is 0, onHit will always return false, so the Ball doesn’t reverse direction, and the score in Board is not changed.

#### boolean onHit(ArrayList<int[]>)

#### Each type of Brick has a different shape, so they determine if they have been hit by the Ball differently. Even though the Ball may be in the same grid coordinate as them, the Ball may have missed them.

#### void onDraw(Canvas)

Draws the brick using shaders to ensure a gradient.

## 2.5 BrickFactory

Makes use of the Factory Method design pattern. It can create different types of Bricks, depending on the BrickType enum passed to it.

### 2.5.1 Structure

#### BrickFactory(Context, BrickType, float, float)

Given the BrickType, Context, and x-y positions, the factory returns a Brick of the specified BrickType.

## 2.6 Paddle

### 2.6.1 Structure

* OnTouch Listener
  + The OnTouch listener is responsible for processing user inputs. We receive user input generated events using the *onTouch()* method and change the position of the Paddle (which is a View) and then redraw it immediately on the screen, effectively moving it
* Hit detection
  + As we get to app completion, we plan to have a function which takes in a Ball object and figures out if it has hit the paddle. And since this function would have access to the Ball, it is able to change the direction of the Ball

## 2.7 Helper

### The Helper class defines static helper methods that are of general enough utility that they do not naturally fall into any of the other classes but can be called and used anywhere in the system. 2.7.1 Structure

* *getDisplayMetrics(), getActionBarHeight(), getStatusBarHeight()* are self-documenting methods that get the relevant height metrics that are mostly used by the Board class to initialize and draw itself
* getNumRows(), getNumCols(), getGridItemSize(), getBoardBoundaries() gets the board dimensions without letting other classes having access to the board object

## 2.8 UML Diagram

# 3.0 General Coding Roles

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Karan Bhandari | Adil Mian | Zeyad Abdulghani | Eric Luo |  |  |  |  |
| Brick(s), BrickFactory | Ball, Helper | Paddle  Connecting two phones | Board, BoardView, GridItem  Image resource files, mockups |  |  |  |  |

Please feel free to view open and closed issues/pull requests on our Git repository to see past and ongoing progress.