+ Game( startingLevel : int )

+ runLevel( numLevel : int ) : void

+ getLevel() : int

+ setLevel( newLevel : int ) : void

+ getScore() : int

+ setScore( newScore : int ) : void

+ getLives() : int

+ setLive( newLives : int ) : void

+ getColor() : Color

+ restartLevel( b : Board ) : void

+ levelScore( timeLeft : int ) : int

-score : int

-frogDirection : frogDirection

-colors : HashMap

-level : int

-lives : int

Class Board

-rows : ArrayList<Row>

-numRows : int

-numFrames : int

-rowSize : int

-frogCurrentRow : int

-frogXIndex : int

-boardScore : int

Class Game

Class Row

+ Board( emptyRowCoef : double, rowSpeedCoef : int, numEnemiesCoef : int )

+ buildBoard( emptyRowCoef : double, rowSpeedCoef : int, numEnemiesCoef : int ) : void

+ boardShift( frogShift : int ) : void

+ drawBoardString( level : int, score : int, lives : int ) : String

+ drawBoard() : void

+ completeBoard() : void

+ moveFrog( moveCode : int ) : void

+ setFrogRow( row : int ) : void

+ setFrogXIndex( x : int ) : void

+ setFrog() : void

+ removeFrog() : void

+ getBoardScore() : int

+ setBoardScore( newScore : int ) : void

-units : ArrayList<Unit>

-rowSpeed : int

-numEnemies : int

-rowSize : int

+ Row()

+ shouldShift( numFrames : int) : boolean

+ rowShift( numFrames : int, frogX : int) : void

+ toString() : String

+ setEnemy( x : int ) : void

+ moveFrogLeftRight( lr : int, frogIndex : int) : void

+ hasEnemy( x : int ) : Boolean

+ frogLeaves( x : int ) : void

+ frogAppears( x : int ) : void

Class Player extends Unit

-sprite : String

+ getSprite() : String

Abstract Class Unit

+ toString() : String

+ abstract getSprite() : String

-sprite : String

+ getSprite() : String

Class Enemy extends Unit

-sprite : String

+ getSprite() : String

Class Ground extends Unit

The two types of collections that we used were Lists and Maps, specifically ArrayList and HashMap. The array lists are found in the Row and Board classes as member variables(list of units and list of rows). The HashMap is a member variable in the Game class, and stores a different color for each key value, which correspond to the game’s different levels.

The program structure is very different from the original UML that was turned in, although a lot of the basic functions and ideas still remain. One difference was that the scope of the project was much larger than we initially anticipated. We needed many more getters and setters for variables like the score, levels, and lives that we didn’t think about beforehand. Another change was getting rid of the weird Queue of numbers of enemies. We decided that it was better to randomly determine the number and speed of the enemies level by level, instead of having them be predetermined. And to be frank, the only reason why the Queue existed in the first place was to have a second type of collection. Thirdly, we converted the program to running in a GUI, and updated the methods drawing the game, and added sprites and the like. However, we still ran it as a text based GUI, so most of the logic surrounding collisions and movement remained the same. Also relating to the Queue change, we added new constructors for the Board class that determined ranges for the row speed, number of enemies, and chance for an empty row based on a bit of math involving the level number. Finally, there were a lot of smaller, fun changes, like adding the different colors for different levels, adding scoring, implementing a GUI, and making a song for the game.