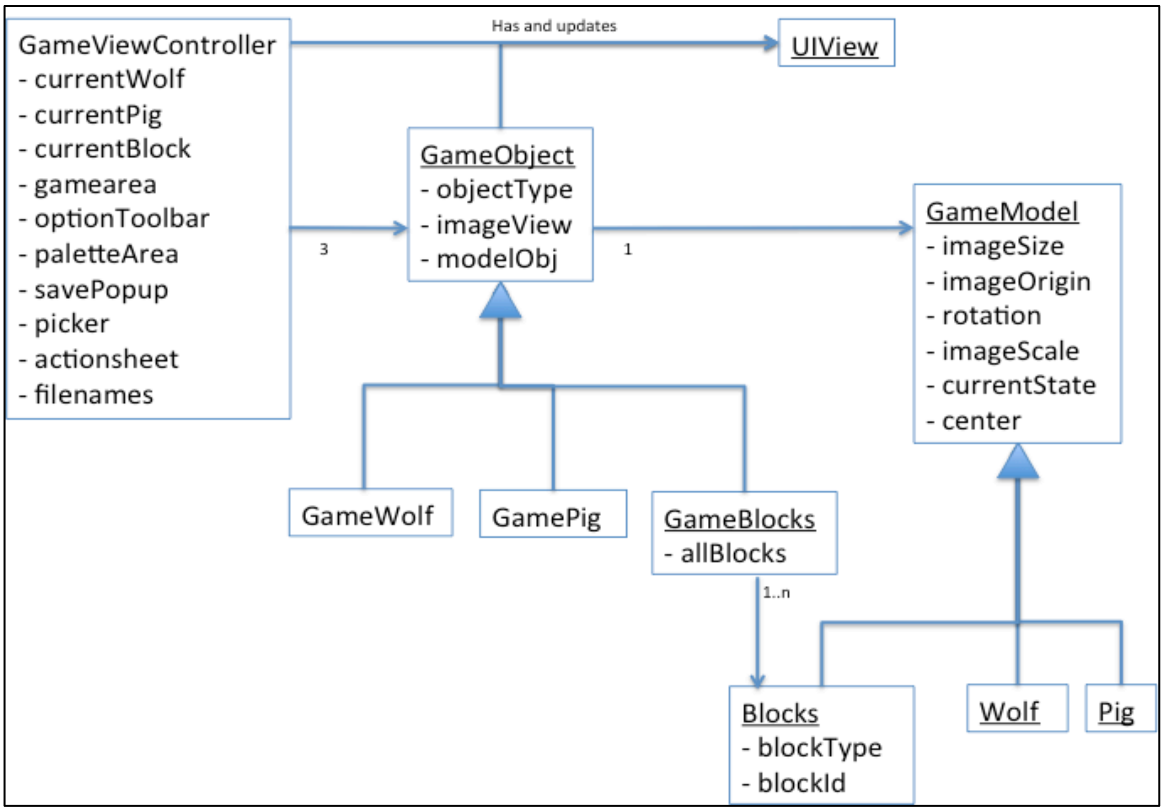
PS3 – Huff and Puff Designer

Problem 2:

a. The concept of Model View Controller is to separate the logic of the application from the interface so that the maintenance of each of the individual components is independent. Hence, I have made certain classes as described in the entity-relation diagram and below that deal with their individual aspects and perform accordingly:



➔ The Model
Classes: *GameModel*, *Wolf*, *Pig*, *Blocks*
Explanation: The model's sole responsibility is to provide information to the view about its state, update its state and notify observers about the updated state. *GameModel* class contains the relevant information about a particular game object (*size*, *origin*, *rotation*, *scale*, *state*, *center*) and has delegates employed to notify the controller when any of these properties are modified. The *Wolf*, *Pig* and *Blocks* are subclasses of this GameModel, where currently *Wolf* and *Pig* are empty classes (will be used in later projects with added properties) and *Blocks* has added properties (*blockType* and *blockId*) and the relevant delegates.
These classes also implement the encoding/decoding for object archive purposes. As these are Model classes, they don't know anything about the controller or the view.

➔ <u>The View</u>

Classes: No additional classes (apart from the ones provided by XCode)

Explanation: Since the view doesn't have any added functionality apart from rendering to the screen, no added classes were explicitly made. The different views present in the application are *gameArea (UIScrollView), paletteArea (UIView), optionToolbar (UIToolbar* and its corresponding buttons*)* and the imageViews of all the sprites in the game. These views are created and modified by the Controller based on the user interaction with the view.

➔ <u>The Controller</u>

Classes: *GameViewController* (& *GameViewControllerExtension*), *GameObject*, *GameWolf*, *GamePig*, *GameBlocks* (& *GameBlocksExtension*)

Explanation: The GameViewController creates the initial background and Palette as soon as the application starts and loads the initializes the objects of the class GameObject by loading the sprites in the palette. Subsequently, it is the GameObject (contains one GameModel object and its imageView) that handles all the gestures, updates the respective view and implements the delegates sent by the Model classes. GameWolf, GamePig and GameBlocks are subclasses to the GameObject. GameWolf and GamePig have very minor added functionality added to the GameObject and will be used more in the future. GameBlocks itself keeps an array of all the blocks present in the game and their corresponding imageViews. GameBlocks has added functionality to add a Blocks model into its array, each time a block is added into the game.
The saving, loading and resetting is handled by the GameViewControllerExtension uses Object Archiver to write/read to the file.

b. Since the application was to be developed based on the MVC pattern, different classes were kept for the Model and Controllers to handle their respective component. This not only ensures independence easy maintenance, but also makes sure that the lowest storage layer knows nothing about the logic or UI layers.
The other possible alternate was to not make separate classes for model and have these Model attributes as part of the Controller properties (similar to the way Apple has implemented UITableViewController by using an array as a model). This would remove the additional layer required to modify each attribute as the Controller can now directly modify its own property. However, the main con is that the Model is obviously not completely separate from the Controller, and hence might prove to be a hassle in future implementations of the physics engine.
Currently the Models also have a delegate, where they assign a delegate each time a modification is made. Similarly, the Controller implements these delegates. This would be of great help in future stages.

c. In order to implement the Wolf's breath, the following steps would be taken:
   - A new Model class, WolfBreath, would be created containing all the necessary projectile attributes (velocity, angle etc)
   - A new Controller class, BreathController, would be created that would contain a WolfBreath object and would update it according to the user interaction with the UI

- WolfBreath would have delegates to inform controllers about the modified state, and the BreathController would implement those delegates to update the View accordingly
- Based on the delegates sent by WolfBreath, the GameWolf will handle the animations of the Wolf (change the image of the wolf)
- Any other controller class, that might be affected by the breath would be modified accordingly

d. In order to integrate the Physics Engine with the Game Objects, the following steps would be taken:
   - The physics engine will be created independently as a separate entity
   - The game Models will have added attributes (mass, inertia, angular velocity)
   - The physics engine will modify the game model's properties according to its own computation
   - The game controllers will have the link with the physics engine, by sending the physics engine its model objects (on which the computations are to be made)
   - The game Controllers will poll the physics engine at a specified rate so that the engine can compute the changes to the model objects (if there are any).
   - At every poll, after the physics engine has modified the game model's properties, the game models send a delegate to the controllers about the updated state
   - The game Controller have already implemented the delegates sent by the models to update the view

e. Apart from the specifications of this problem set, other features that could be added to the designer stage are:
   - The ability to change the background of the gamearea
   - Implementing some animations to game objects when they are idle in gamearea (like basic jumping, or round robin with some expressions)
   - Giving more characters than just wolf and pig
   - Deleting previously saved levels as the list might increase after certain time
   - Ability to load saved files by pressing next (so that user can easily navigate through saved levels)
   - Not limiting the current gamearea to the given image specification, but being able to zoom out of the gamearea to design a large scale level

   Out of the additional features, the two that I would prioritize are:
   - Option of deleting previously saved levels
     When the user presses the load button, and the list of save files pops up, there will be a button to delete to selected name. This will be implemented by adding a new button in the toolbar of the pop list of save games, called Delete. The target method of this button would be created in the GameViewControllerExtension that would handle the pressing of delete, and would delete the file from system based on the selected name.

No major change in class structure, some additional functionality in the GameViewController and its extension class.
If required, a Delete All can be implemented in a similar fashion.

- The ability to change the background of the gamearea & animal characters
  A gesture like swipe would be defined for the UIScrollView gamearea and added to it. This will be done in the GameViewController, which would handle the swipe gesture by changing the imagview of the background and replace it with something else.
  The other animal characters would be similar to the round robin of blocks, where tapping an animal would change it to another. This would be done in GameWolf and GamePig classes by adding certain methods (like GameBlocks). Also, the class name may be changed to more like GameKiller, GameVictim as GameWolf and GamePig becomes too specifc

Problem 5
Since this application could not be tested through unit tests, it was broken down into smaller components and tested manually as described below:

Black-box testing:
- On application first-time launch:
  - The wolf, pig and straw-block object should be present on the palette
  - The gamearea should be vacant
  - Reset button pressed - shouldn't have any impact on the layout
  - Save button pressed – prompt for input name and shows the message whether the file was successfully saved
  - Load button pressed – display no saved files alert message
- On palette:
  - Rotate the object in place – the object should not rotate
  - Pinch the object in place – the object should not resize
  - Tap on the object in place – the block should change type in the order straw-wood-stone-iron, wolf object and pig object should not respond to this gesture
  - Double tap on the object in place – the objects should not respond to this gesture
  - Drag the object in place – as soon as the drag starts, the object should resize and drop into the game area (without any rotation)
- Transition from palette to gamearea:
  - Drag from palette to gamearea – the object should retain its original default size and shouldn't have any rotation
  - The wolf and pig object should no longer be present on the palette
  - If a block is dragged to the game area, a new straw block should appear on the palette (that conforms to all the conditions of "on palette" mentioned above)
  - As soon as the object is dragged, the object should stay within the bounds of the gamearea
- In gamearea:

  - Translate the object in place – the respective object should translate based on the finger location, but would get blocked near the boundaries of the game area (cannot cross the game area in any direction) – This should remain valid even if the object is rotated or scaled
  - Translate the object in place – while the user is dragging the object in the gamearea, the gamearea should not scroll
    For blocks, attempt to drag should not change block type
  - Pinch the object in place – the respective object should resize according to the pinch amount (in the scale range of 0.6 to 1.6)
  - Rotate the object in place – the respective object should rotate according to the amount of rotation placed on them
  - Rotate and Pinch simultaneously – both operations should be performed simultaneously, with the object being resized and rotated
  - Single Tap the object in place – only block should respond to this gesture by changing its type in the round robin fashion as mentioned previously
  - Double tap object in place – the object should be deleted from the gamearea; if the object is a wolf/pig, a new wolf /pig should appear on the palette.
  - Dragging on the plain scrollable area should simply scroll the gamearea
  - Rotation, pinch, double tap or tap on plain scrollable area should not have any impact on the gamearea
  - Performing any of the gestures within the pixel range of the sprite – the sprite should respond instantaneously, and any action even close to the sprite should not have any impact on the object sprite
- Save button:
  - Since the game is always in designer mode at the moment, it should pop an alert view whenever the button is pressed prompting the user to enter the name of the file to be saved.
  - If the user inputs a file name that is already present, it will overwrite an existing file (maybe warn the user in future implementations that such a file already exists)
  - If the user enters an invalid name, an error message will popup
- Load button:
  - It should display a list of saved files to choose from – selecting one name should load the saved design immediately and close the list
  - If there are no saved files, then an alertview message will show
- Reset button:
  - As soon as the reset button is pressed, all the objects return to their natural state and all the “on palette” rules described above apply
  - The functionality of the reset button should be valid at any point in the game
  - Since there is no undo feature, once it is reset, the user cannot go back to the previous state (unless it was saved)
- Application switched
  - The game state should resume from the last state when the application was switched to some other application

Since the internal methods were either gesture specific, view related or setter/getters of model objects, other forms of testing were not implemented