PS3

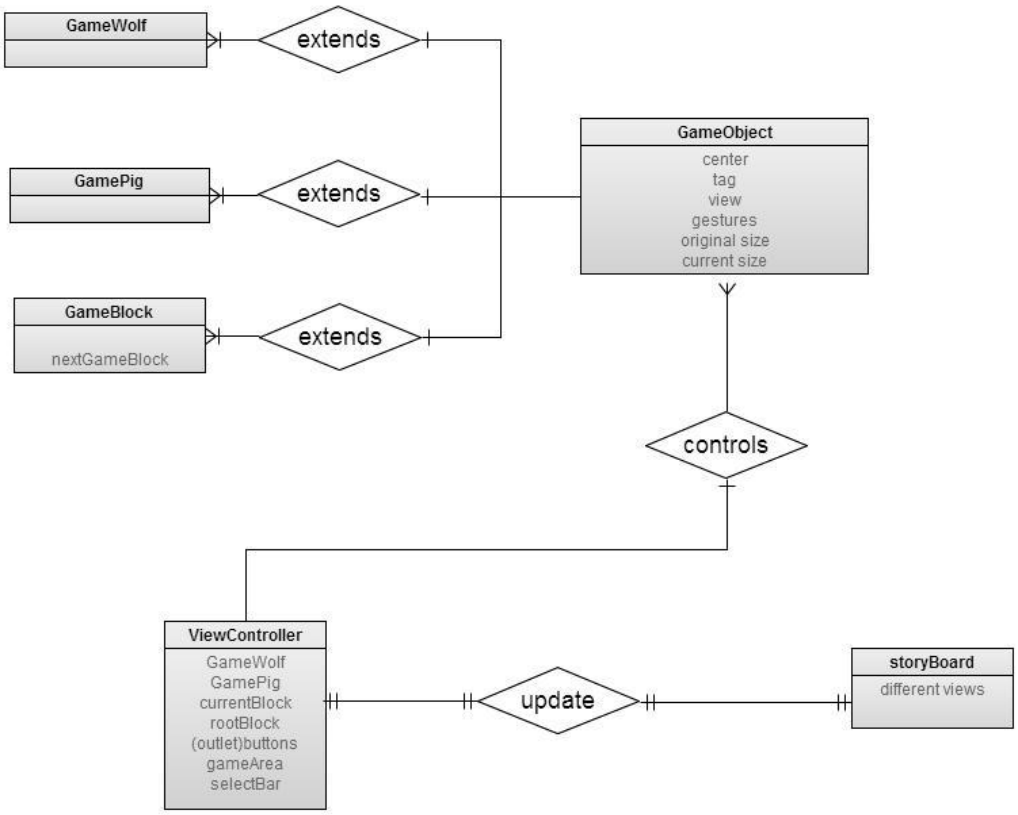
Cui Wei A0091621M

Problem 2:

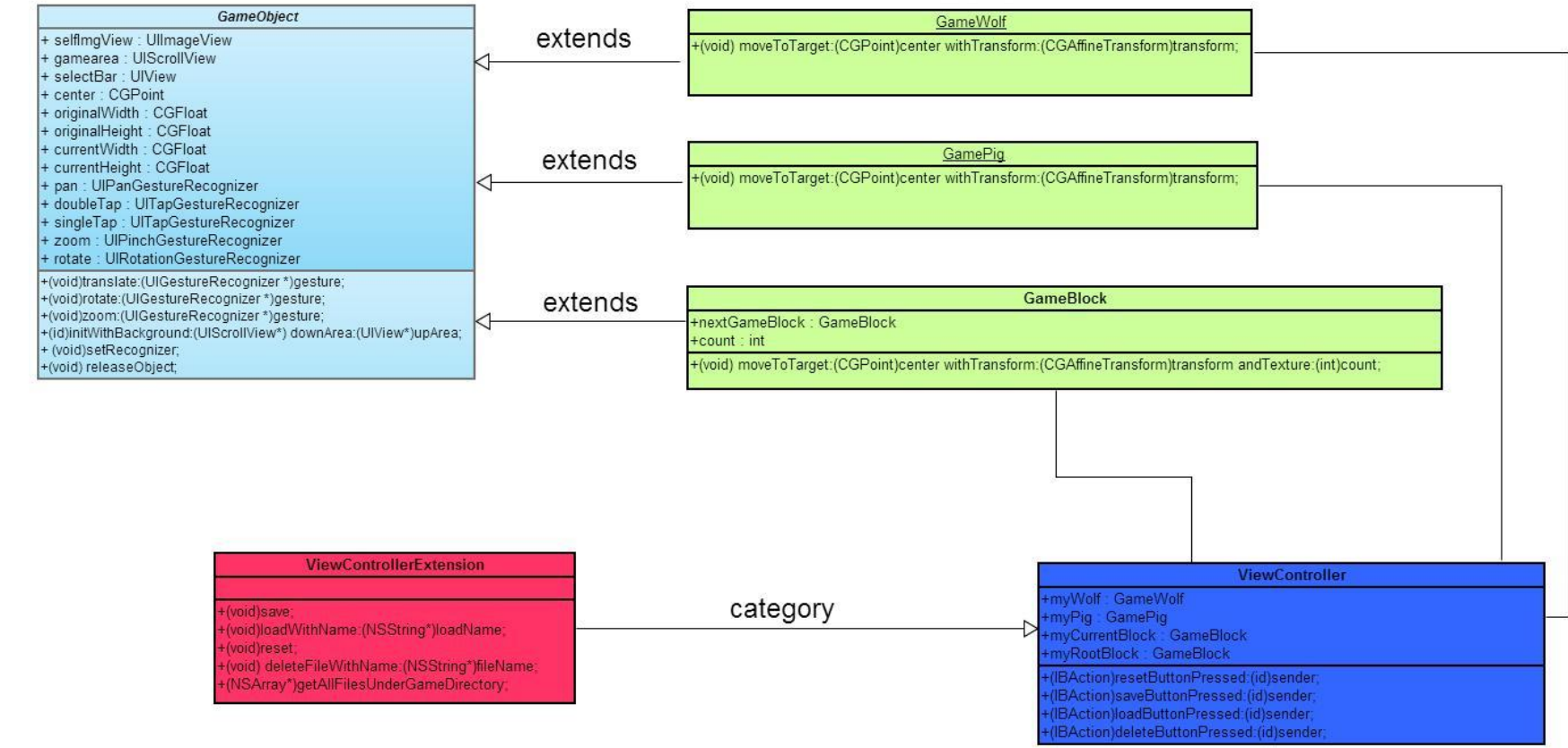
(a).

The view is simply the storyBoard and the viewController is just the control. The model is combined in the gameObject(it is the gameObject's properties). So when I have to save the game state, I just need to save these properties to remember the game state.

Entity Relationship Diagram (ERD)



Class Diagram



(b)

Because it is most easily to understand and to implement. Also it satisfy the requirement of MVC, and ensures code reusability. Extending existing class(GameObject) can reduce the number of codes, and since there is only one scene at all times, one MVC, thus one view and one controller is sufficient enough.

(c)

A breadth is just another game object. So I would just add it like GameWolf, GamePig or GameBlock. The thing to note is that I will not add it as an ivar of GameWolf. GameWolf and Breadth will be coordinated in terms of position in the view by controller, so they behave like one is linked with another.

(d)

Physical engine is just another class, or most probably, classes. It controls the position of all the GameObjects. The communication between different objects will be done by delegate, so that controller can update them periodically.

(e)

Just like the real angry birds, the breadth can be in different form (not just air, but also fire to burn the woods, and ice to freeze iron so it can be broke easily with next breadth). The changes now, given the MVC pattern, can be easily made, by simply add some more possible views for each object, and some adjust in the physical engine.

Problem 4

My way of implementing save and load function is manual coding. I store the vital properties of each objects(its view, actually) into a dictionary and store the dictionary into the document directory. This is a very simple and effective way of storing files. Since We do not need to store large amount of data, we do not need to use Core Data. And a dictionary has its built-in functions to be stores in the flash drive. Also, the saved file is a XML style file thus can be easily verified during testing and debugging phase.

The load function is also very simple, since we can easily extract information from a dictionary into an array, and re-construct game state with the stored info.

Problem 5

Black-Box testing

-test gameobjects: *drag views but not to the gameArea

*drag views to gameArea

*reize, rotate and move views in gameArea

*drag views out of the bounds of gameArea

*double tap to reset a view

*reset the whole view

*scroll view can be scrolled at all scenarios

-test file operation: *save file with different game state (no change, some changes, extensive changes)

*load different file

*delete different file

Glass-Box testing

The glass-box testing is similar to the above testing hierarchy. However, this time I will use NSLog function to log various information.

for example :

-test gameobjects: *drag views but not to the gameArea

NSLog(view.superView == selectBar)

NSLog(view.center)

...

-test file operation: *save file with different game state (no change, some changes, extensive changes)

*load different file

*delete different file

NSLog (@"%@", [fileManager contentsOfDirectoryAtPath]) for each file operatin

Bonus Question:

1. This time I can safely say, I spent more than 40 hours on this problem set.

2. Things would be much easier if I could be more familiar with the APIs I need to use.

3. I really hope the problem set can contain more detailed instructions for the possible APIs that we might need.