CS3217 Problem Set 3

Name: Tay Yang Shun

Problem 2a

I applied MVC in the following way:

Model

The data containing the GameObjects (Wolf, Pig and Blocks) are attributes in GameViewController class.

GameObject has the following attributes which define the state of the object that is common to all 3 subclasses:

- **Game Object Type:** Defines the kind of object (Wolf/Pig/Block).
- **Origin:** The origin point of the view frame in its unrotated state. The origin is defined as the top left hand corner of the rectangle frame.
- **Center Point in palette:** The center point of the view frame. This is useful in defining the location of a rotated object because it remains unchanged regardless of the rotation.
- **Game Object View:** Essentially the image view of the GameObject. The class' view frame has the same frame as this image.
- **Rotation State:** The angle in which the frame has undergone.
- **Game State:** Boolean attribute which indicates the location of the view (palette or gamearea)

The 3 classes have further attributes:

<u>GameWolf and GamePig</u>

- **Initialization State:** The default initializer will make the wolf and pig icons appear in the palette at their respective locations.
- **Custom initializers:** This custom initializer is used for displaying the wolf/pig in non-default positions. It will take in a CGFrame define the view size and to pinpoint the location of the image view before rotation, the rotated angle, and also the game state of the view (whether inside palette or gamearea) and subsequently display the wolf/pig.

<u>GameBlock</u>

- **Initialization State:** The default initializer will make the block icon appear in the palette at its respective locations.

- **Custom initializers:** This custom initializer is used for displaying the block in non-default positions. It will take in a CGFrame define the view size and to pinpoint the location of the image view before rotation, the rotated angle, and also the game state of the view (whether inside palette or gamearea) and subsequently display the block in the correct orientation.
- **Array of different types of Game Blocks:** Stores an array consisting of the images of the different kinds of blocks. To be referred to when changing block types.
- **Game Block Type:** The current material (type) of the block.

View

The view contains three main subviews: menu bar, palette and gamearea.

GameWolf and GamePig

The GameWolf and GamePig subviews can exist either in palette or gamearea and not both. There can be any positive number of blocks existing at any one time. When the GameWolf/GamePig gets dragged from the palette into gamearea, they are transferred into the gamearea with the view resized to the proper dimensions. When a double tap for these two objects occur, they are removed from gamearea will be returned to the palette. GameWolf/GamePig objects are resizable and rotatable.

GameBlock

GameBlock subviews can appear concurrently in the palette and gamearea. To make things simple, There will be only one block instance permanently in the palette. When a block gets dragged into the gamearea from the palette, a new block will appear in the palette so that the user can drag more blocks into the gamearea if he desires to. When a double tap for a block occurs, it is simply removed from view and not returned to the palette. A tap changes the type of block to be made of different materials. GameBlock objects are resizable and rotatable.

Menu Bar

- **Save:** Saves the state of the wolf, pig and array of blocks in the gamearea.
- **Load:** Load the state of the wolf, pig and array of blocks in the gamearea.
- **Reset:** Returns the wolf and pig back to the palette. All blocks in gamearea are removed.
- **Level:** The string in this field indicates the level name to be saved into/loaded from when the save/load button is pressed.

Controller

The main controller is the GameViewController. The main attributes of the class are:

- **Wolf Controller:** A GameWolf object that has the wolf image view and its attributes. It initializes the wolf image view based on the frame, rotation and game state of the wolf.
- **Pig Controller:** A GamePig object that has the pig image view and its attributes. It initializes the pig image view based on the frame, rotation and game state of the pig.
- **Block Controller:** A GameBlock object that has the block image view and its attributes. This controller will permanently be controlling the block that appears in the palette.
- **Array of GameBlock Objects:** This array will store the GameBlock objects that appear in the game area and their respective states.
- **File Data Manager:** Class that manages the saving and loading of the level. Encodes and decodes the properties of the GameObjects into an NSKeyedArchiver/NSKeyedUnarchiver.

Delegates are set up between between the GameViewController and the GameObjects to serve the following purposes:

- When a GameObject is being translated around in the gamearea, a message is sent from the object to the main controller to disable the scrolling of the gamearea.
- When GameObject is dragged physically out of the palette, a message is sent from the object to the main controller to remove the object from the palette and create a new object of the right size at the new location in gamearea. If the GameObject is of the block type, a new GameBlock is created in the palette.
- When a GameObject experiences a double tap, a message is sent from the object to the main controller to remove the object from the gamearea. If the GameObject is not of block type, a GameObject of the respective kind will be created in the palette for subsequent usage.

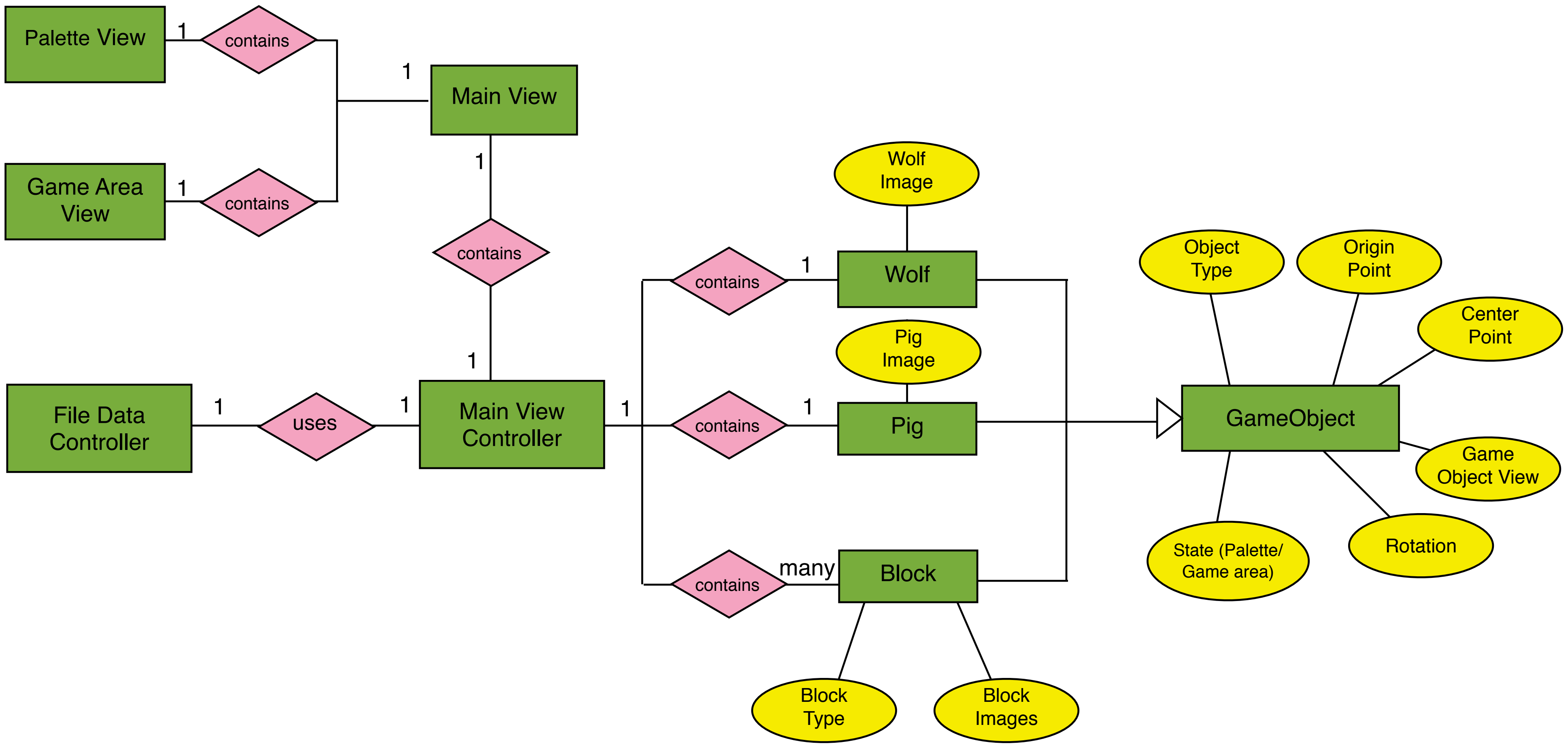
Saving and loading is done via communication between the GameViewController and the FileDataController.

When the save button is pressed, the important attributes of the GameObjects in the palette and gamearea are archived in the file determined by the level name string in the text field. When the load button is pressed, the required attributes are unarchived from the file determined by the level name string in the text field and new instances of the GameObjects are created depending on the state and attributes defining it.

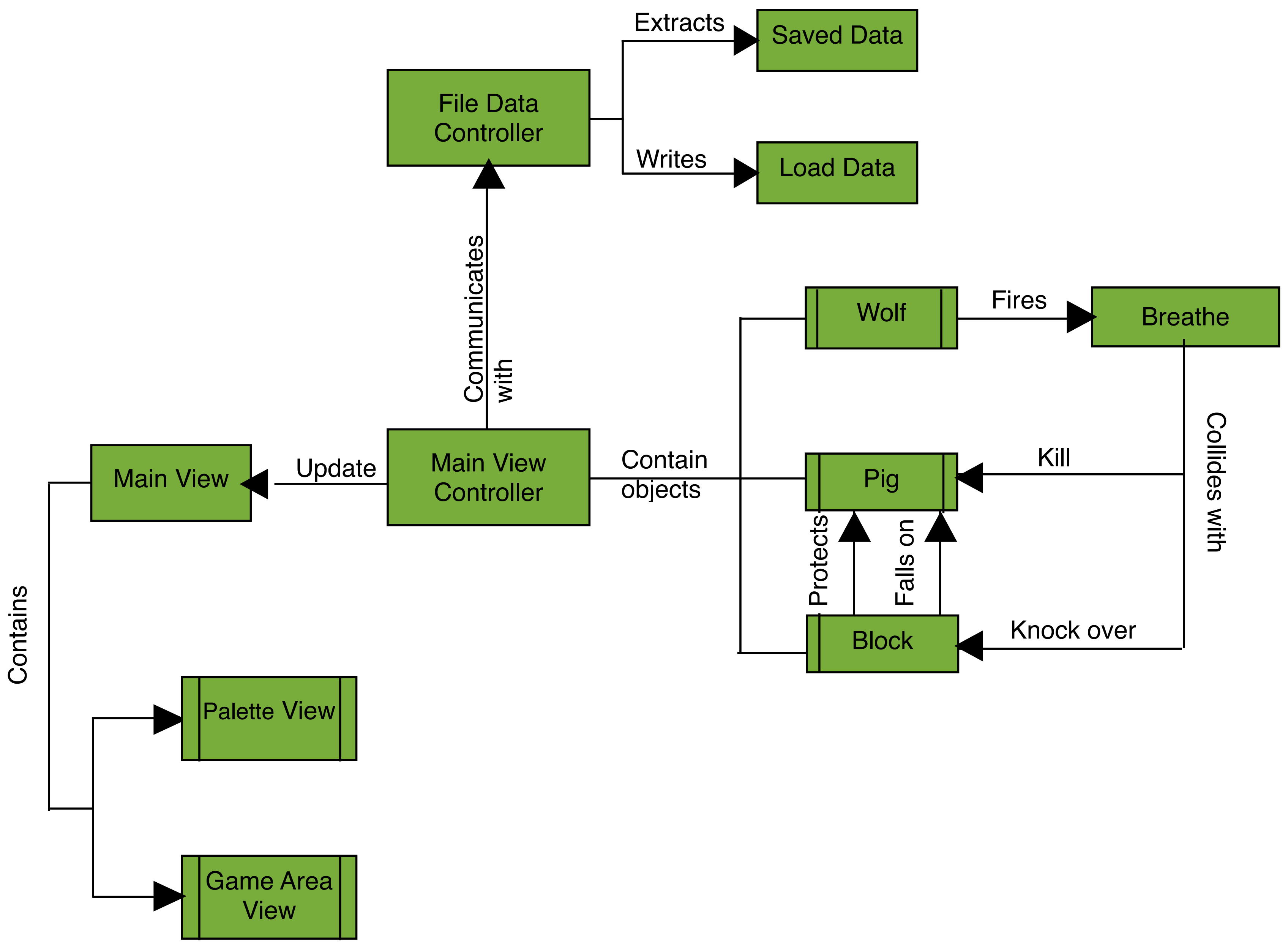
The reset button basically removes all objects in the gamearea and restores the initial state of the palette.

Gestures are detected by the views and the controller will process these gestures by sending messages to the model (state of the objects) and modifying it.

Entity-Relationship Diagram



Module-Dependence Diagram



Problem 2b

I decided to use the GameViewController as the central controller so that it will communicate between the view and the model (game objects). As **all** messages have to be passed via the GameViewController, I believe this design is simple as easy to follow. The GameViewController will handle all communication between the view and the model. There will not be alternative paths of communications between the model and the view. I chose this design as the GameViewController observes the view and will have control over all the data. It will be hence able to make the suitable decision based on the view's input through its observation.

Problem 2c

I will add it as a new GameObject subclass called GameBreath, which contains the breath image. When a long press gesture is detected on the wolf, the GameBreath object is created at the GameWolf's mouth. It will travel in a projectile motion after the press is lifted, depending on the length of time pressed and the angle.

Problem 2d

I will add a few attributes to each GameObject:
• CGFloat mass
• CGPoint velocity (point is used because the velocity is represented as a 2-d vector)
• CGFloat rotational velocity

I will implement a method that checks for overlapping of objects and using the physics engine equations for collisions, modifies the attributes of the respective objects. The method will be called at a 1/60 seconds interval so that the movement will appear smooth.

Problem 2e

1) Different coloured wolves. It will be similar to how I implemented the block images changing function. An array of UIImages storing the images of the different coloured wolves will be saved in the GameWolf object. When a single tap gesture is experienced, the image of the wolf will be changed.

2) Background change. Depending on the time the player is playing the game, the background of the gamearea can be changed. I have made a night version of the gamearea background. In the loadView method, NSDate will be use to retrieve the current time of play and the relevant background will be loaded.
3) Multiple pigs. Similar method has been implemented to enable having many blocks in the gamearea. The objective of the game will then be to kill all the pigs.

Problem 4

I created a UITextField object and placed it beside a label called "Level". The text entered in that text field indicates the level name to be saved/loaded. I created a custom class called FileDataController which communicates with the GameViewController in handling the saving and loading of GameObject attributes.

<u>Saving</u>

When the save button is pressed, the FileDataController object retrieves the string currently in the text field and checks if there is an existing level with such a name. If there is, it then prompts the user to make a decision whether to overwrite that file. After which, the attributes of the GameWolf, GamePig and GameBlocks are encoded and saved in an NSKeyedArchiver. The important attributes to be saved are:

GameWolf: Frame, rotation and game state
GamePig: Frame, rotation and game state

The frame, rotation and game state is enough to define the position of the wolf/pig in the gamearea.

GameBlocks array: Number of blocks in the gamearea, each block's individual frame, rotation and block type.

Because of the way I structured my program, there will permanently be a GameBlock icon appearing in the palette. Hence there is no need to save it in the object archives. Regarding the array of blocks, the frame and rotation of the block is sufficient to define a rotated state of a block of any size, while block type is used to store the kind of block it is.

Loading

When the load button is pressed, the FileDataController object retrieves the string currently in the text field and checks if there is an existing level with such a name. If there is no such level, an alert will be displayed, indicating “No such level found”. If there is such a level, the saved attributes are decoded using an NSKeyedUnarchiver and new instances of the GameWolf, GamePig and GameBlocks will be created in their respective locations followed by displaying them on the main view.

I did not save the UIImages of the wolf, pig and blocks because they are stored in their own object class I can retrieve them when using the custom initializers.

Choice of Maintaining Data Persistence

The important attributes that I want to save are: View frame, Game state, Rotation and Block type. However, view frames which are of CGRect type cannot be saved in a property list. Hence I chose to use Object Archives because it is more flexible than property lists and it can archive CGRects. I archived these GameWolf, GamePig and GameBlock properties individually. In future versions where the GameObjects will be more sophisticated and there are more properties, I can create larger custom game object classes to store one whole object in one go.

Problem 5

Black box testing:

Function	Wolf	Pig	Block
Drag within palette (Pan Gesture)	Return back to starting point	Return back to starting point	Return back to starting point
Drag from palette into gamearea (Pan Gesture)	Image of bigger dimensions created	Image of bigger dimensions created	Image of bigger dimensions created
Drag object within gamearea (Pan Gesture)	Object moves with finger	Object moves with finger	Object moves with finger
Resize object (Pinch Gesture)	Resized according to distance between fingers	Resized according to distance between fingers	Resized according to distance between fingers
Rotate object (Rotate Gesture)	Rotated about the center point according to finger positions	Rotated about the center point according to finger positions	Rotated about the center point according to finger positions
Drag multiple objects into gamearea, with that object already in gamearea (Pan Gesture)	Palette is empty	Palette is empty	Multiple objects can be dragged into the gamearea
Change object image (Tap Gesture)	Unable to change	Unable to change	Changes block image according to order defined
Remove from gamearea (Double Tap Gesture)	Moved back into palette	Moved back into palette	Removed from gamearea
Reset button with all objects in gamearea	Moved back into palette	Moved back into palette	All blocks removed from gamearea
Reset button with all objects in palette	No change	No change	No change

Glass-box testing:

Function	Effect	Result
With objects in place, save to new file titled “Level1”. Reset button pressed. Load file with same title.	Game objects returned to their position in the gamearea	Pass
With objects in new positions, save to existing file titled “Level1”. “Yes” button pressed when asked if user wants to overwrite.	Overwrites existing save file with the name “Level1”. Positions are changed according to the new save file state.	Pass
With objects in new positions, save to existing file titled “Level1”. “No” button pressed when asked if user wants to overwrite.	Existing save file not overridden. Position of game objects are loaded from the original save file.	Pass
With objects in place, save to new file titled “Level2”. Reset button pressed. Load “Level2” file.	Game objects returned to their position saved in the file “Level2”.	Pass
With objects in place, save to new file titled “Level2”. Reset button pressed. Load “Level1” file.	Game objects returned to their position saved in the file “Level1”.	Pass
Close the app and reopen it. Load “Level1” file.	Game objects returned to their position saved in the file “Level1”.	Pass
Close the app and reopen it. Load “Level2” file.	Game objects returned to their position saved in the file “Level2”.	Pass

Bonus Problem: Reflection

a.
I spent probably 60 hours. This is due to the large amount of time spent on designing the implementation and making sure that it will work in the subsequent parts. I started last Friday and spent 5 days figuring out the proper way to use the GameObject view controllers. I couldn’t figure out why the image did not appear when I added it as a subview. I think this problem set allowed me to learn LOADS about GUI programming. I had no prior experience with GUI programming and the learning curve was extremely steep. Thanks to online references and help from my fellow classmates, I managed to pull through this problem set. I really put in a lot of effort into this problem set.

b.
I could have discussed with my peers regarding the approach they took for this problem set. I was stuck at the first part for 5 days and after I figured that part out, progressed from 5% completion to 70% completion within a night. The initial part was extremely demoralizing.

c.
If possible, please provide *a little* more guidelines for this problem set. I know that finding our own way of implementing the program is part and parcel of the learning, but it would really help many people if we were steered in the right direction from the start. There are very few examples ViewController subclasses being used as objects in the view and quite a few of us spent a large amount of time finding the right direction to start.