Sprint 3

MouseController.cs

Code is perfectly readable and easy to understand. I would recommend adding more comments, specifically before each method that briefly describes the functions of the method. Since this is a relatively small and simple class, this isn't really an issue, but more comments would make the class more convenient to read. Ideally, the comments should be able to allow somebody who has no idea how this class works to use this class quickly and precisely.

Also recommend that these are separated out as function calls:

```
37          public void Update(GameTime gametime)
38          {
39
40              previousMouseState = currentMouseState;
41              currentMouseState = Mouse.GetState();
42              foreach (var mappedState in mouseMappings)
43              {
44
45                  if (mappedState.Item2 == 0)
46                  {
47                      // check previous state to run only once on button press
48                      if (currentMouseState.LeftButton == ButtonState.Pressed &&
49                      {
50                          mappedState.Item1.Execute();
51                      }
52
53                  }
54                  if (mappedState.Item2 == 1)
55                  {
56                      // check previous state to run only once on button press
57                      if (currentMouseState.RightButton == ButtonState.Pressed &&
58                      {
59                          mappedState.Item1.Execute();
60                      }
61
62                  }
63              }
64          }
```

A context switch occurs inside this iterator, thus moving this code inside a function call is recommended. It seems this is the case for KeyboardController as well.

Not sure why there is a Draw method inside this class.

```
public void Draw(GameTime gameTime)
{
    //not implemented
}
```

It seems that KeyboardController also has an unimplemented Draw method. Should remove this dead code from these classes and the IController interface.

Aside from this, the code is of high quality and appears to be simple to extend if more functionality is needed.

RoomObject.cs
Code throughout the file is very readable and simple to understand,
However there appears to be some cohesivity issues.

The enemy AI logic is done inside of RoomObject instead of owned by the enemy.

```
27        //enemy AI related data
28        private List<SpriteAction> enemyActions;
29        private SpriteAction enemyAction;
30        private Random rand;
```

**This is just the variables for the enemy AI that is implemented inside this class's Update() method.*
Should move this functionality into an AI class that is owned by each enemy object.

Collision detection code needs to be moved to a new class that handles detecting collisions for relevant game objects. Needs to be moved from RoomObject, this class should not be handling iterating through colliders.

```
98          //update Link
99          if (Link != null) {
100             Link.Update(gameTime);
101             //TODO: move collision updates into its own manager class
102             ((IConcreteSprite)Link).UpdateCollideWithWall(this);
103             if (Link.collider.isIntersecting(RoomObjectManager.Instance
104                 Link.collider.isIntersecting(RoomObjectManager.Instance
105             {
106                 //TODO: link takes damage
107                 TakeDamage(Link);
108             }
109         }
```

TakeDamage() method should be a command instead of a method inside RoomObject. Needs to be moved from RoomObject, this class should not have code relating to entities taking damage. Also should consider renaming that variable.

```
270     private void TakeDamage(ISprite sprite)
271     {
272         IConcreteSprite castSprite = (IConcreteSprite)sprite;
273         SpriteAction newPos;
274         float orgX;
275         float orgY;
276
277         /* Decrement the sprites health field */
278         castSprite.health--;
279
280         /* Keep the sprite facing in the same direction when they take
```

The code inside TakeDamage() could also do with some refactoring and comments.

```
282             switch (spritePos)
283             {
284                 case 0:
285                     newPos = SpriteAction.damageLeft;
286                     orgX = castSprite.screenCord.X;
287                     orgY = castSprite.screenCord.Y;
288                     castSprite.screenCord = new Vector2((orgX + 20), orgY);
289                     break;
290                 case 1:
291                     newPos = SpriteAction.damageRight;
292                     orgX = castSprite.screenCord.X;
293                     orgY = castSprite.screenCord.Y;
294                     castSprite.screenCord = new Vector2((orgX - 20), orgY);
295                     break;
296                 case 2:
297                     newPos = SpriteAction.damageUp;
298                     orgX = castSprite.screenCord.X;
299                     orgY = castSprite.screenCord.Y;
300                     castSprite.screenCord = new Vector2(orgX, (orgY + 20));
301                     break;
302                 case 3:
```