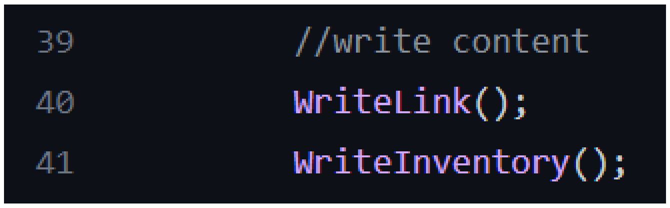
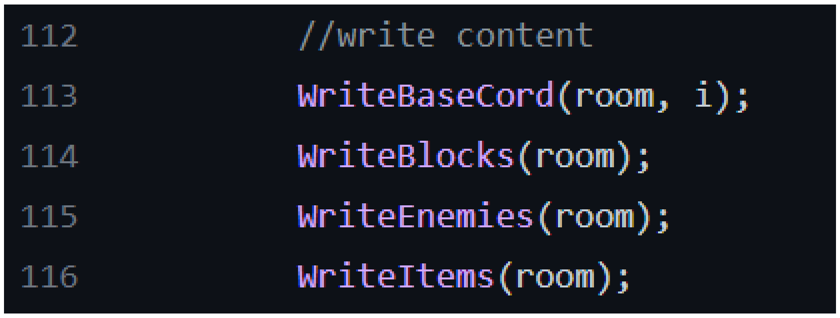
Sprint 5

LevelSaver.cs
Most of this file is well-organized, well-implemented, and intuitive to understand. The public methods make strong usage of private method calls to handle contextually separated aspects of the whole function.



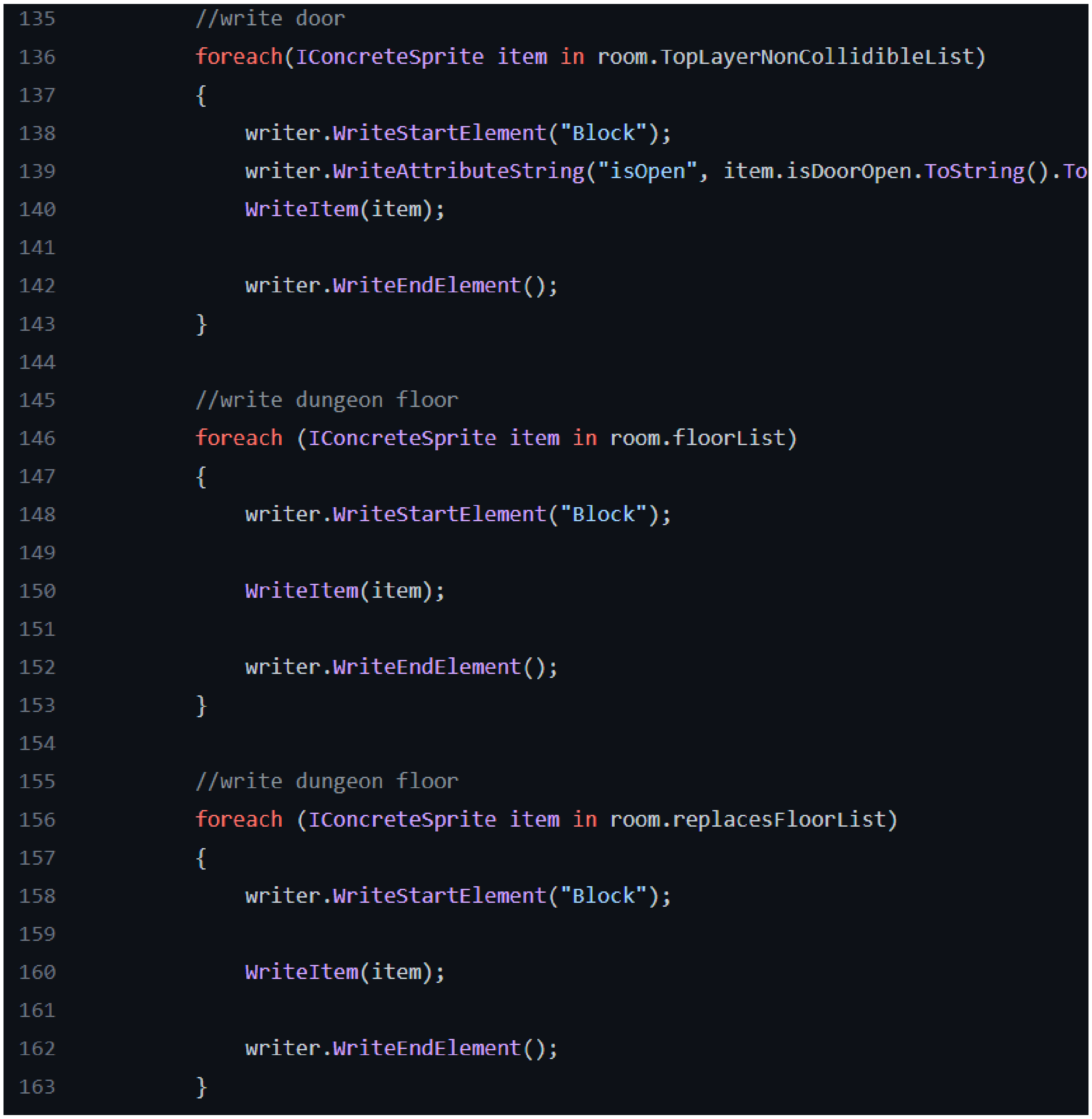
```
        //write content
        WriteLink();
        WriteInventory();
```



```
        //write content
        WriteBaseCord(room, i);
        WriteBlocks(room);
        WriteEnemies(room);
        WriteItems(room);
```

The decision to make the ‘writer’ variable a class variable was also ideal as most of the functions would have required a writer, and this method is simpler than passing the writer as an argument.

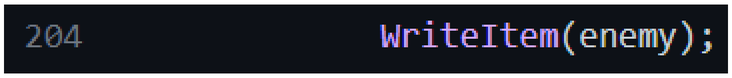
The WriteBlocks() method can be said to be too long, as it doesn't fit on one screen.



```
        //write door
        foreach(IConcreteSprite item in room.TopLayerNonCollidibleList)
        {
            writer.WriteStartElement("Block");
            writer.WriteAttributeString("isOpen", item.isDoorOpen.ToString().To
            WriteItem(item);

            writer.WriteEndElement();
        }

        //write dungeon floor
        foreach (IConcreteSprite item in room.floorList)
        {
            writer.WriteStartElement("Block");

            WriteItem(item);

            writer.WriteEndElement();
        }

        //write dungeon floor
        foreach (IConcreteSprite item in room.replacesFloorList)
        {
            writer.WriteStartElement("Block");

            WriteItem(item);

            writer.WriteEndElement();
        }
```

Each of these foreach loops can be moved out as a separate private method call instead, which will significantly reduce the height of the WriteBlocks() method.

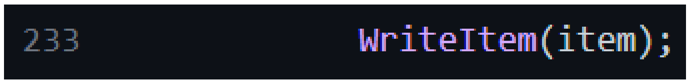
Misleading use of the term "Item" in this context. In the repository, "Item" refers to the class hierarchy that encompasses projectile entities and item drops, but this method name uses "Item" to refer to a single element in a list or set.



```
            WriteItem(enemy);
```

This method handles enemies, which do not fall under the "Item" class hierarchy.

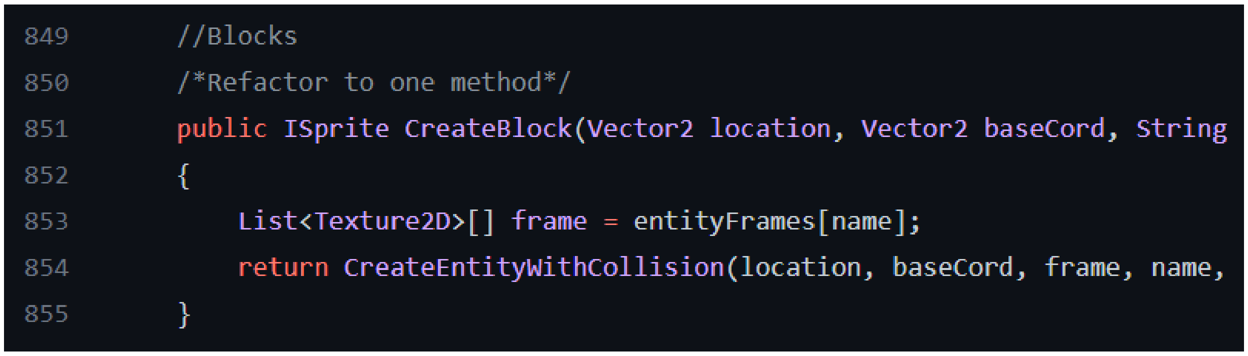
On the other hand, there is a similar overload of the WriteItem() method that handles objects under the “Item” class hierarchy.



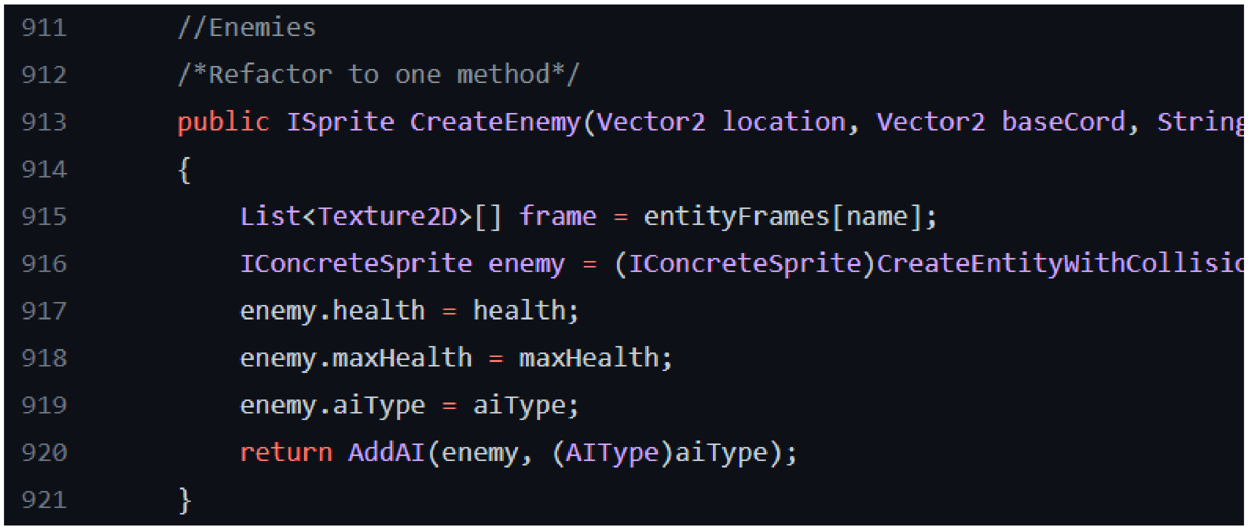
```
                WriteItem(item);
```

SpriteFactory.cs
I have written about SpriteFactory in S4, but I’ve been told that it's been refactored. As such, this review won’t address the existing issues already covered inside the S4 review, but will instead cover the changes. If it is not addressed later, assume that that aspect of SpriteFactory has not changed.

Most of the methods that were called to create a specific entity have been massively reduced in number. Now there is generally only one method for each category of entity. There are now significantly less violations of DRY.



```
        //Blocks
        /*Refactor to one method*/
        public ISprite CreateBlock(Vector2 location, Vector2 baseCord, String
        {
            List<Texture2D>[] frame = entityFrames[name];
            return CreateEntityWithCollision(location, baseCord, frame, name,
        }
```



```
        //Enemies
        /*Refactor to one method*/
        public ISprite CreateEnemy(Vector2 location, Vector2 baseCord, String
        {
            List<Texture2D>[] frame = entityFrames[name];
            IConcreteSprite enemy = (IConcreteSprite)CreateEntityWithCollisio
            enemy.health = health;
            enemy.maxHealth = maxHealth;
            enemy.aiType = aiType;
            return AddAI(enemy, (AIType)aiType);
        }
```

The only problem now is that these generalized methods usually have quite a few arguments in order to handle all the different entities that it needs to build. There does not seem to be a feasible solution for this without refactoring over half of existing code..