Sprint 4

SpriteFactory.cs

The SpriteFactory has seen significant growth as more features are added to the game. Unfortunately, it has now become heavily bloated and very difficult to refactor.

The SpriteFactory still does not return Sprites objects, it returns entities objects, and the entity object is the one that holds pointers to the frames, which are all initialized and stored within SpriteFactory.

Currently, the SpriteFactory class handles creating all of the game objects needed by LevelLoader when parsing the room xml files.

The recommended course of action is to separate out the Sprite creation aspect of SpriteFactory, and rename this class to GameObjectBuilder.

1. This class primarily builds all of the game objects in the rooms, and has private methods that add Collider objects and AI objects to the game objects.
2. Remove the 150+ lists of frames and lists of lists of frames. Refactor to have a new, actual, SpriteFactory2 load in frames into Sprite objects, and have that stored in a dictionary.
3. Instead of adding Lists directly into code, have SpriteFactory2 load in Sprites by parsing an xml.
4. This also involves refactoring ConcreteSprite (the entity class) into using a pointer to the Sprite object.

Doing this should clear up ~600 lines of repetitive list declaration and initialization code. Essentially, turn all of this into xml entries:

```
12      private List<Texture2D>[] barrierFrames;
13      private List<Texture2D> barrier;
14      private List<Texture2D>[] stairsFrames;
15      private List<Texture2D> stairs;
16      private List<Texture2D>[] waterFrames;
17      private List<Texture2D> water;
18      private List<Texture2D>[] roughFloorFrames;
19      private List<Texture2D> roughFloor;
20      private List<Texture2D>[] dungeonFloorFrames;
21      private List<Texture2D> dungeonFloor;
22      private List<Texture2D>[] alternateBackgroundFrames;
23      private List<Texture2D> alternateBackground;
24      private List<Texture2D>[] invisibleBarrierFrames;
25      private List<Texture2D> invisibleBarrier;
26      private List<Texture2D>[] openingFrames;
27      private List<Texture2D> opening;
28      private List<Texture2D>[] textFrames;
29      private List<Texture2D> text;
```

```
201         textFrames = new List<Texture2D>[4];
202         barrierFrames = new List<Texture2D>[4];
203         stairsFrames = new List<Texture2D>[4];
204         waterFrames = new List<Texture2D>[4];
205         statueLeftFrames = new List<Texture2D>[4];
206         statueRightFrames = new List<Texture2D>[4];
207         roughFloorFrames = new List<Texture2D>[4];
208         dungeonFloorFrames = new List<Texture2D>[4];
209         alternateBackgroundFrames = new List<Texture2D>[4];
210         invisibleBarrierFrames = new List<Texture2D>[4];
211         openingFrames = new List<Texture2D>[4];
```

```
385         text.Add(content.Load<Texture2D>("DungeonSprites/Text"));
386         opening.Add(content.Load<Texture2D>("BlockSprites/Black"));
387         invisibleBarrier.Add(content.Load<Texture2D>("BlockSprites/InvisibleBarrier"));
388         alternateBackground.Add(content.Load<Texture2D>("DungeonSprites/Room07"));
389         dungeonFloor.Add(content.Load<Texture2D>("DungeonSprites/DungeonFloor"));
390         barrier.Add(content.Load<Texture2D>("BlockSprites/Barrier"));
391         stairs.Add(content.Load<Texture2D>("BlockSprites/DungeonStairs"));
392         water.Add(content.Load<Texture2D>("BlockSprites/Water"));
393         roughFloor.Add(content.Load<Texture2D>("BlockSprites/RoughFloor"));
394         statueRight.Add(content.Load<Texture2D>("BlockSprites/StatueRight"));
395         statueLeft.Add(content.Load<Texture2D>("BlockSprites/StatueLeft"));
```

(there is a lot more)

Next, as stated prior, SpriteFactory currently returns game objects in response to LevelLoader calls. Currently, there is one method for each game object, and thus dozens of methods of repetitive code that can be made into parameters for much fewer generalized methods. Unfortunately, LevelLoader uses these methods, so refactoring would involve changing LevelLoader as well.

SoundFactory.cs

Stores a dictionary of all sounds and returns them. Solid class. Highly reusable and extendable. There is some dead commented code here and there but the class's simplicity facilitates no issues with readability.

That said, should ideally load all sounds by parsing a file, instead of inside the code. Similar problem as SpriteFactory.

```
28          soundEffects.Add("Dungeon 1", Content.Load<SoundEffect>("Sound2/Dungeon 1"));//done, tested
29          soundEffects.Add("LOZ_Arrow_Boomerang", Content.Load<SoundEffect>("Sound2/LOZ_Arrow_Boomerang"));//done, tested
30          soundEffects.Add("LOZ_Bomb_Blow", Content.Load<SoundEffect>("Sound2/LOZ_Bomb_Blow"));//done tested
31          soundEffects.Add("LOZ_Bomb_Drop", Content.Load<SoundEffect>("Sound2/LOZ_Bomb_Drop"));
32          soundEffects.Add("LOZ_Boss_Hit", Content.Load<SoundEffect>("Sound2/LOZ_Boss_Hit"));//done tested
33          soundEffects.Add("LOZ_Boss_Scream1", Content.Load<SoundEffect>("Sound2/LOZ_Boss_Scream1"));
34          soundEffects.Add("LOZ_Door_Unlock", Content.Load<SoundEffect>("Sound2/LOZ_Door_Unlock"));//done, tested
35          soundEffects.Add("LOZ_Enemy_Die", Content.Load<SoundEffect>("Sound2/LOZ_Enemy_Die"));//done tested
```

SoundManager.cs

Could do some comments explaining exactly what each method does, but clean class regardless.

There appears to be some very specific code relating to Link's hurt and sword sounds. This probably shouldn't be here.

```
76      /*
77       * This holds the complexity for the sounds of attacking with sword,
78       * taking damage, opening/closing doors
79       *
80       * What is a sprite action?
81       */
82      public void playStateSounds(SpriteAction action, ISpriteState state)
83      {
84          switch (state.toString())
85          {
86              case "AttackState":
87                  SoundManager.Instance.PlayOnce("LOZ_Sword_Slash");
88                  break;
89
90              case "DamagedState":
91                  SoundManager.Instance.PlayOnce("LOZ_Link_Hurt");
92                  break;
93
94
95          }
96      }
```

We really should have made a class for each separate game object type :(