Code Smells Focused On

- Application-level
    - Duplicated Code
    - Shotgun Surgery
- Class-level
    - Data clump
- Method-level
    - Excessively long line of code

- **Interfaces**
    - ICollectable
        - Has similarities to a statemachine, do we do a collectable statemachine?
    - ICollideable
        - SLIPING TO NEXT SPRINT
    - ICommand
        - N/A
    - IController
        - For the “RegisterCommand” function, I’m not exactly sure what the parameter “Rectangle position” is used for, add comments for parameters for clarity?
    - IDrawable
        - Comment for “IsDrawn” should be ‘Reports whether <c> this </c> is drawn to the screen or not’?
        - For the “ToggleDrawing” function, what/where is it toggling?
    - IEnemy
        - Could change function name “Dead” to “Die”, but not of great importance, understanding of functions still makes sense
    - IInteractable
        - INTERACTIONS NOT SETUP YET, WILL BE DONE IF TIME
    - IItem
        - N/A
    - IMoveStateMachine
        - N/A
    - IPlayer
        - N/A
    - IProjectile
        - N/A
    - ISprite
        - N/A
    - IStructure
        - The interface seems a little bare right now, is there anything else a structure would need functionality-wise besides “GetHitBox” and a Sprite?
    - IStructureStateMachine

        - Not all structures can be destroyed, so the “Break” and “Destroy” functions would be a bit extra for some objects.
    - IThinkingStateMachine
        - N/A
    - IUpdateable
        - N/A
- **Classes**
    - BackgroundObject
        - Shouldn’t need to implement IItem (As the comment is asking)
        - If the object gets an ISprite object, then it shouldn’t need Columns, Rows, SourceRectangle, or Texture variables
        - We may need a BackgroundStateMachine for when we implement scrolling into the game
        - Also shouldn’t need MarioPosition
        - Not sure about graphics variable
        - The constructor needs a lot of parameters here, may be reduced when/if variables listed above get removed/changed
        - The rest will get changed/updated when we start implementing background objects into the game
    - BigMarioDecorator
        - In “Update” function, instead of checking for StateMachine.states[“Shrunk”], check for “Damaged” and StateMachine.health < 2. And call StateMachine.Shrunk().
        - I think that the MarioDecorator class in the same file as BigMarioDecorator is just supposed to be the Mario class that is in it’s own separate file already (Look at the Damaged Link decorator example on Kirby’s website)
    - Block
        - N/A
    - BlockStateMachine
        - N/A
    - BlockSwitchingCommand
        - The switching commands are all very similar, but it’s not hugely important for this sprint.
    - EnemySwitchingCommand
        - See BlockSwitchingCommand
    - FireballStateMachine
        - Similar to ShellStateMachine, can/should we connect in some way?
    - FireballUpdate
        - Should not need a Sprite variable. The fireball class will hold this.
    - FireMarioDecorator
        - In “Update” function, instead of checking for StateMachine.states[“LoseFire”], check for “Damaged” and StateMachine.health < 3. And call StateMachine.LoseFire().

        - I think that the MarioDecorator class in the same file as FireMarioDecorator is just supposed to be the Mario class that is in it's own separate file already (Look at the Damaged Link decorator example on Kirby's website)
        - Extra note, just to make sure we know, when Mario takes damage with Fire power, he still remains Big Mario. So the decorators should be layered in a way that looks like FireFlower(BigMario(SmallMario())) and have to remove one at a time
    - Flag
        - N/A
    - FlagStateMachine
        - Shouldn't have to worry too much until a later sprint.
    - GameObjectManager
        - May need a backgroundList when we get to background objects.
        - We do not need a Piranha enemy for level 1 as far as I know
        - The StateMachines for most of the Objects should have a Reset() function that resets the initial position already, as well as their states. They also have their very initial positions hardcoded in there. Do we want the GameObjectManager to send the initial conditions to the statemachines for this sprint?
        - When creating the objects, we need to make sure their constructors get passed the correct things they need.
    - GameState
        - Very long, may want to consider breaking this up a bit in some way if possible.
        - Shouldn't have to worry too much about this until the next sprint.
    - Goomba
        - N/A
    - GoombaStateMachine
        - Could remove "FaceRight" or "FaceLeft" state, as one could already tell us the implied state of the other.
    - GoombaUpdate
        - N/A
    - ItemStateMachine
        - Could remove "FaceRight" or "FaceLeft" state, as one could already tell us the implied state of the other.
    - ItemSwitchingCommand
        - See BlockSwitchingCommand
    - ItemUpdate
        - N/A
    - KeyboardController
        - N/A
    - Koopa
        - N/A

- KoopaStateMachine
  - Could remove “FaceRight” or “FaceLeft” state, as one could already tell us the implied state of the other.
- KoopaUpdate
  - N/A
- LittleMarioDecorator
  - In “Update” function, instead of removing the decorator, it should be adding the BigMario decorator if Mario has grown. And knowing that Mario has grown isn’t the StateMachine’s job, the StateMachine should actually be getting told about Mario growing itself.
  - The MarioDecorator class issue is the same as the other decorators
- Mario
  - Mario only needs 1 StateMachine, there are 2 that are instantiated though
  - Something just doesn’t seem right about Mario’s functions having only one line of code, and they all are calls to StateMachine
  - “Jump” function should call StateMachine.Jumping(GameTime gameTime) instead of “InAir” and send that function gameTime
  - “TakeDamage” function should send StateMachine.Damaged() gameTime
  - If GameObjectManger is calling Mario.Update() then Mario is going to have to have an update function, even if that means it’s just calling statemachine.Update(gameTime);
- MarioJumpCommand
  - If the space bar is held down, will that continue to execute this command continuously? Or will it activate only once? This affects the statemachine and how it updates Mario’s position on screen.
- MarioLeftCommand
  - See MarioJumpCommand
- MarioRightCommand
  - See MarioJumpCommand
- MarioStateMachine
  - Could remove “FaceRight” or “FaceLeft” state, as one could already tell us the implied state of the other.
  - A bit on the long side, can we compress or split in some way?
- MarioThrowingCommand
  - See MarioJumpCommand
- MouseController
  - N/A
- Pipe
  - N/A
- PipeStateMachine
  - Shouldn’t have to worry too much until a later sprint.
- QuitGameCommand
  - N/A
- Shell

        - The speed shouldn't be needed here. The statemachine will handle that, so it should be calling statemachine functions instead of setting values on its own for a speed inside itself.
    - ShellStateMachine
        - See FireballStateMachine above.
    - ShellUpdate
        - N/A
    - Sprite
        - Is it at all possible to combine the 2nd and 3rd constructors into 1 with the same parameters as the 2nd? And to hold up what the 3rd constructor does we would just have to send in SpriteEffects.None into that last parameter.
    - SpriteFactory
        - Quite a long class, is there any way we could simplify/shorten it?
        - I would think changing "HitBlock" to something like "UsedBlock", "FinishedBlock", or "DoneBlock" would help with clarity.
    - StarMarioDecorator
        - In the "Update" function, losing the star depends on time. And the decorator should be tracking this. Then when time runs out, it call StateMachine.LoseStar() so the statemachine knows Mario no longer has the star.
        - MarioDecorator class issue is the same as the above decorators' issues.
    - Text
        - N/A
    - ResetCommand
        - I would assume this is not finished yet because the GameObjectManager did not have a way to reset yet?

Summary of Code as Whole

For upcoming work, we need to focus on connections between the Updates and States, and the Commands and the GameObjectManager and GameState. The GameObjectManager has a good set connection with the Game Objects already which is nice. There does seem to be some uncertainty still in how Updates, actual Game Objects, their States, and any Decorators will share responsibility in controlling the inner workings of the objects.