Amos Pivato, Matthew Hackenbrook, Nabil Haque, Jathuzan Kulasekaram

Design Patterns

Creational design patterns

- Singleton Design Pattern
    - Classes have one instance and have global points of access.
    - Examples:
        - The WorldHandler is a class used to implement the game and is the global point of access for running the game.
        - The DataManager which is used to manage the input and output of the game data.
        - The WorldAI class is used to generate valid NPC commands at random to allow them to interact with the game world.
    - This design pattern was chosen because it provides a central location for one instance of this class.

Structural design patterns

- Adapter Design Pattern
    - parserCommand works as an adapter to change the user input into commands that can be used to action events and activities in the game
    - User inputs are taken as strings and converted to the Command Model format

Behavioral design patterns

- State
    - Our NPCs in the game have states.
    - For example we have an Idle state, Wander state, and a Hostile State
        - While in wander the character can wander around, unlock doors, and take items.
        - While in Hostile they will be able to attack. They only enter this state when there is another character present in the room that is hostile to them.
        - While in the idle state they just stay where they are and don’t execute any commands.
    - State changes are randomized internally everytime the WorldAI class is initialized

Amos Pivato, Matthew Hackenbrook, Nabil Haque, Jathuzan Kulasekaram

AntiPatterns

Software Development AntiPatterns

- The Blob -
    - The Blob antipattern contains a majority of the process.
    - Refactoring - we avoid The Blob by not having one giant class for the entire game and by creating separate classes to handle tasks and relieve the central WorldHandler class.
- Lava Flow-
    - We had designed dead code in our data file that we had initially planned to use but later decided to not use.
    - Refactoring - We took out the dead code from the data file

- Input Kludge-
    - Our ParserCommand class was falling for most imputed commands due to the mishandling of the inputs.
    - Refactoring - we removed the Input Kludge by specifying more detailed conditions in the parser and the validator for the inputs that were being received.

Project Management AntiPatterns

- Throw it over the Wall-
    - We were following our class diagram to a fault and ended up creating classes for modules that didn't need them
    - Refactoring - we turned the classes into method only modules to be imported when needed.