Adventure in the Neighborhood – Project To-Do List

1. Project Reflection & Self-Grading (Required at the End)

Part 1: Response to Feedback

- [ ] Copy and paste the **three reflection questions** into your submission
- [ ] Answer **Question 1 (Strengths)**
  - Choose one strength from teacher feedback
  - Explain **why it worked well**
  - State whether it was easy or challenging
- [ ] Answer **Question 2 (Areas for Improvement)**
  - Choose the **most significant issue**
  - Explain **what code or logic you would change** (no full rewrite required)
- [ ] Answer **Question 3 (Push Opportunity)**
  - Describe **how you would implement the extension**
  - List **new variables, methods, or classes**

Part 2: Self-Grading

- [ ] Review the **rubric categories**
- [ ] Assign yourself a score out of **200 points**
- [ ] Write **2–3 honest sentences** justifying your score
- [ ] Fill in:
  My Self-Assessed Score: ___ / 200

---

2. Game World & Grid System

- [ ] Create a **2D grid** (5x5 or 7x7 array)
- [ ] Decide what each cell can contain:
  - [ ] Buildings
  - [ ] Obstacles
  - [ ] Items
  - [ ] NPCs
  - [ ] Empty spaces
- [ ] Track the **player's position**
- [ ] Prevent movement outside the grid or into walls

2. Game World & Grid System

- [ ] Create a **2D grid** (5x5 or 7x7 array)
- [x] Decide what each cell can contain:
  - [ ] Buildings
  - [ ] Obstacles
  - [x] Items
  - [x] NPCs

    - [x] Empty spaces
- [x] Track the **player's position**
- [x] Prevent movement outside the grid or into walls (bounds checking implemented)

---

3. Classes & Inheritance

Required Class Structure

- [ ] Create a **Location superclass**
- [ ] Create subclasses of Location:
    - [ ] Building
    - [ ] Park
    - [ ] Shop
- [ ] Create an **Item superclass**
- [ ] Create Item subclasses:
    - [ ] Food
    - [ ] Tool
    - [ ] Treasure
- [ ] Create a **Character superclass**
- [ ] Create subclasses:
    - [ ] Player
    - [ ] NonPlayerCharacter (NPC)

3. Classes & Inheritance

Required Class Structure

- [x] Create a **Location superclass**
- [ ] Create subclasses of Location:
    - [ ] Building
    - [ ] Park
    - [ ] Shop
- [ ] Create an **Item superclass**
- [ ] Create Item subclasses:
    - [ ] Food
    - [ ] Tool
    - [ ] Treasure
- [x] Create a **Character superclass**
- [x] Create subclasses:
    - [x] Player
    - [x] NonPlayerCharacter (NPC)

---

4. Variables, Encapsulation & Constructors

Encapsulation

- [ ] Make all instance variables **private**

- [ ] Create **getters and setters** for key variables

Attributes to Include

- [ ] Character:
  - name
  - health
  - inventory
  - position
- [ ] Item:
  - name
  - description
  - value

Constructors

- [ ] Add **no-argument constructors**
- [ ] Add **parameterized constructors**
- [ ] Use the `this` keyword correctly
- [ ] Use `super` where appropriate

4. Variables, Encapsulation & Constructors

- [x] Make all instance variables **private**
- [x] Create **getters and setters** for key variables

Attributes to Include

- [x] Character: name (present)
- [ ] Character: health (implemented on `Player`)
- [x] Character: position (present)
- [x] Player: health & inventory present on `Player`

Item attributes

- [ ] Item: name, description, value (project has `InteractiveItem` but no general `Item` with `value`)

Constructors

- [x] Add **no-argument constructors** (many classes include them)
- [x] Add **parameterized constructors** (present where used)
- [ ] Use `super` where appropriate (some uses present; may need refactor for new hierarchies)

---

5. Methods & Game Behavior

Player Methods

- [ ] `move(direction)`
- [ ] `pickUpItem(item)`

- [ ] `useItem(item)`
- [ ] `talkTo(npc)`

Location / Building Methods

- [ ] `enter(player)`
- [ ] `exit(player)`

Shop Methods

- [ ] `buyItem(player)`
- [ ] `sellItem(player)`

Output Methods

- [ ] Override `toString()` in at least one class
- [ ] Use `toString()` to display object details

5. Methods & Game Behavior

Player Methods

- [ ] `move(direction)` (movement implemented via `moveNorth`/`moveSouth`/etc.; consider centralizing)
- [x] `pickUpItem(item)` (pick up handled in game loop)
- [x] `useItem(item)` (basic `use` handling in game loop)
- [ ] `talkTo(npc)` (not implemented as `Player` method)

Location / Building Methods

- [ ] `enter(player)`
- [ ] `exit(player)`

Shop Methods

- [ ] `buyItem(player)`
- [ ] `sellItem(player)`

Output Methods

- [x] Override `toString()` in at least one class (implemented in `Location`)
- [x] Use `toString()` to display object details

---

6. Control Structures & Game Logic

- [ ] Use `if / else if / else` for:
  - Invalid movement
  - Health reaching zero
  - Interactions
- [ ] Use loops (`while` or `for`) for:

    - Repeated player turns
    - Game progression
- [ ] Define at least one **win or end condition**

6. Control Structures & Game Logic

- [x] Use `if / else if / else` for invalid movement, interactions (implemented via switch/if logic)
- [x] Use loops (`while`) for repeated player turns and game progression
- [x] Define at least one **win or end condition** (lose condition: health <= 0 implemented)

---

7. Randomization (Math Usage)

- [ ] Use `Math.random()` for:
    - Random events
    - Random item placement
    - Optional encounters

7. Randomization (Math Usage)

- [x] Use `Random`/randomization for NPC placement and random events (implemented)

---

8. User Input & Interaction

- [ ] Use a `Scanner` to read user input
- [ ] Allow commands such as:
    - `move north`, `move south`, `move east`, `move west`
    - Short commands (e.g., `m`)
- [ ] Validate user input
- [ ] Provide feedback for invalid commands

8. User Input & Interaction

- [x] Use a `Scanner` to read user input
- [x] Allow commands such as `move north`, `move south`, `move east`, `move west`
- [ ] Short command aliases (e.g., `m`) not implemented
- [ ] Input validation can be improved (basic handling exists)

---

9. String Manipulation

- [ ] Use string concatenation for output
- [ ] Use `equals()` for command comparison
- [ ] Use formatting methods (`substring`, etc.)
- [ ] (Optional) Create a **text-based puzzle** using strings

9. String Manipulation

- [x] Use string concatenation for output

- [x] Use `String`-based command comparison (switch on `String` used)
- [ ] More formatting methods and an optional text puzzle not implemented

---

10. Data Storage & Object Interaction

- [ ] Store inventory in an `ArrayList` or array
- [ ] Loop through collections to:
  - Display inventory
  - Use or remove items
- [ ] Demonstrate object interactions:
  - Player ↔ Item
  - Player ↔ NPC
  - Player ↔ Building
- [ ] Include at least one interaction that **changes game state**

10. Data Storage & Object Interaction

- [x] Store inventory in an `ArrayList` (`Player.satchelItems`)
- [ ] Explicit loops to display inventory are not used (ArrayList.toString() printed instead)
- [x] Demonstrate Player ↔ NPC interactions (NPC encounters change health)
- [ ] Player ↔ Building interactions not implemented
- [x] Include interactions that change game state (health, items picked up)

---

11. Output, Polish & Presentation

- [ ] Clean, readable output formatting
- [ ] Clear instructions for player commands
- [ ] Descriptive messages for actions
- [ ] Organized and readable code

11. Output, Polish & Presentation

- [ ] Output is functional but could use polishing (help text exists)
- [x] Clear instructions for commands present via `help()`
- [x] Descriptive messages for many actions present
- [ ] Further code cleanup and organization suggested

---

12. Design & Documentation (20 pts)

- [ ] Create class diagrams or written class descriptions
- [ ] Explain class interactions
- [ ] Provide an overall game design plan

12. Design & Documentation (20 pts)

- [x] Class diagram added in `diagrams/classes.puml` (update after design changes as needed)

- [ ] Written class interaction explanations and overall design plan need expansion