# School Simulation Game – Project Plan

## Project Overview and Goals

The goal of this project is to develop a **2D school-life simulation game engine** focused on student NPC (non-player character) behavior and player exploration. We aim to simulate a living school environment where NPC students follow daily routines (e.g. attending classes, visiting dorms) in real-time, creating an immersive world that feels alive and dynamic[[1]](https://playgama.com/blog/general/how-can-i-design-npc-schedule-mechanics-like-those-in-stardew-valley-for-caroline-to-enhance-player-interaction-in-my-game/#:~:text=NPC%20schedule%20mechanics%20are%20crucial,similar%20to%20Stardew%20Valley%E2%80%99s%20Caroline). The player controls a student character (PC) who can freely move around the campus map and observe or eventually interact with these NPCs. By focusing on core systems like movement and scheduling first, we will lay a robust foundation for future content (like dialogues, quests, or detailed graphics) once the engine is proven stable.

**Key Objectives:**

* **Core Engine Focus:** Emphasize gameplay mechanics (movement, scheduling, AI behavior) over artwork or story in initial development. We will use simple placeholder graphics for characters and environments to accelerate prototyping[[2]](https://blog.lipsumhub.com/placeholder-content-in-game-development/#:~:text=The%20use%20of%20placeholder%20,in%20creating%20the%20final%20assets). This allows us to refine movement and AI without getting bogged down in asset creation. All placeholder assets will be clearly labeled for easy replacement with final art later.
* **Believable NPC Behavior:** Implement a scheduling system so each NPC follows a believable daily routine. NPCs should attend classes, take breaks, return to dorms, etc., according to a time-of-day cycle. This will make NPCs act as independent, self-directed characters rather than static figures, enhancing realism and player immersion[[1]](https://playgama.com/blog/general/how-can-i-design-npc-schedule-mechanics-like-those-in-stardew-valley-for-caroline-to-enhance-player-interaction-in-my-game/#:~:text=NPC%20schedule%20mechanics%20are%20crucial,similar%20to%20Stardew%20Valley%E2%80%99s%20Caroline). Variation will be added so routines aren’t perfectly repetitive (for example, occasional random activities or slight schedule shifts) to mimic real-life behavior.
* **Explorable Campus Map:** Create a basic campus map (classrooms, dormitory, common areas, etc.) using placeholder graphics. The map will be **extensible**, meaning new locations (buildings, rooms) can be added over time without major changes to the code. All game action will occur within this map – NPCs won’t “teleport” or exist off-screen. This constraint simplifies the simulation and ensures the player can witness all NPC activities in the game world.
* **Player Navigation and Agency:** Enable the player-controlled character to move throughout the map using keyboard input (e.g. WASD or arrow keys). Movement should feel smooth and responsive, allowing the player to explore all accessible areas. The player can roam the campus to follow NPCs or go anywhere of interest. While interactive features (talking to NPCs, picking up items) will be **added in later phases**, the initial focus is simply moving around and observing the environment.
* **Foundation for Expansion:** Design the engine with future expansion in mind. New NPCs, new schedules, and new locations should be straightforward to introduce (data-driven where possible). For example, adding a new student with a unique schedule or a new building on the map should not require rewriting core systems. This flexibility in design will accommodate growing the simulation’s scope in subsequent updates.

## Scope and Approach

**Project Scope:** In this phase of development, we are concerned with implementing the **movement and scheduling systems** using placeholder content. Art, storylines, and complex interactions are **out of scope** for now. The scope includes: creating a placeholder-based map environment, one playable character with movement mechanics, a set of NPCs with timed movement routines, and the underlying time/schedule manager. Everything will be kept simple and testable. We assume a top-down 2D perspective (think classic RPG style) for movement on the map. All character sprites and tiles will be basic placeholders (e.g. colored squares or simple pixel icons) – sufficient to recognize entities and test collisions, but not final quality. The game will run on a simulated day cycle (e.g. a looping 24-hour timeline or a simplified schedule cycle) to drive NPC routines.

**Development Approach:** We will follow an iterative, milestone-based approach to build and verify each core feature incrementally. Each milestone delivers a **distinct, testable feature** (e.g. player movement, then NPC movement, etc.), which must meet its acceptance criteria before we proceed to the next stage. This gated progression ensures we catch issues early and have a working game at each step. By breaking the project into clear milestones, we maintain focus and can validate one aspect of the engine at a time.

Our strategy is to **prototype with placeholders** and refine gameplay mechanics first. Using placeholder graphics and dummy content is a conscious choice to prioritize core functionality – it lets us focus on how movement feels and how AI behaves without waiting on artwork[[2]](https://blog.lipsumhub.com/placeholder-content-in-game-development/#:~:text=The%20use%20of%20placeholder%20,in%20creating%20the%20final%20assets). For example, rather than designing detailed classroom art now, we might use a simple grid or silhouette for a classroom area labeled "Classroom A". This practice ensures progress isn’t stalled by asset production and helps expose any gameplay issues early when they are cheaper to fix.

**NPC Scheduling System:** A critical part of the approach is designing the NPC scheduling system. We will create a global clock/timer in the game that simulates the passing of a day. NPCs will have data defining their schedule (e.g. NPC1: 8:00 AM go to Classroom; 12:00 PM go to Cafeteria; 5:00 PM return Dorm). At each time trigger, the NPC’s AI will update its target location and move there via a pathfinding mechanism (for simplicity, possibly a grid-based path or predefined waypoints). Initially, schedules can be simple and synchronized (all classes start at say 8 AM and end at 12 PM). Once that works, we will introduce variations: different class times for different NPCs, random idle movements when they have free time, etc. The NPC movement algorithm will ensure they navigate around obstacles/walls and do not walk through each other if possible. Since all NPC action stays on the single map, we do not need to simulate any off-screen behavior – if an NPC is scheduled to “go home” outside campus, for now they might just walk to a map edge or a dorm building on the map and stop there.

**Extensibility Considerations:** From the start, we will structure data and code to allow easy expansion. For instance, NPC schedules will be defined in external data structures (like a JSON or scriptable objects) so new NPC routines can be added without code changes. The map will be built in a modular way – possibly divided into sections (e.g. by building) – so we can append or swap in new sections. We will keep placeholder assets for any new area ready so that adding a new location is as simple as drawing a new rectangle or loading a new dummy image and updating the navigation logic. By the final milestone, we intend to **demonstrate extensibility** by adding at least one new location and NPC to prove the concept.

Throughout development, we will **test each component** thoroughly. For movement, we’ll test boundary conditions (the PC cannot walk through walls or out of bounds, NPCs properly follow paths). For schedules, we’ll run the simulation through a full day to ensure NPCs end up where expected at the right times and handle transitions properly. We’ll also check that multiple NPCs don’t conflict (e.g. all crowding into the same spot). Each milestone’s completion will be verified with specific test scenarios (detailed in the acceptance criteria below and in the delegation matrix). Only when a milestone’s deliverables meet the criteria do we move on – this guarantees stability and that we always have a working build.

Finally, clear documentation and labeling will accompany all placeholder elements. The codebase will include comments or a guide indicating where placeholders are used (so artists/developers know what to replace later). For example, if a sprite named npc\_placeholder.png is used for all NPCs now, we’ll note that this is temporary. Similarly, any hard-coded test behavior will be marked. This avoids confusion down the line and ensures a smooth transition when integrating final assets or more complex features.

## High-Level Acceptance Criteria

The following are the high-level acceptance criteria defining when the core engine is considered successful. These criteria are phrased broadly; each milestone in the Delegation Matrix will have more detailed, specific acceptance checks for that phase. At a minimum, the project will be considered *complete for this phase* when all of these are true:

* **Player Free Movement:** The player’s character can be controlled to move around all intended areas of the map without glitches. Movement should respect the environment (no walking through walls or obstacles) and feel responsive to input. The PC can reach each key location on the map (e.g. each classroom, dorm room, common area) via some path.
* **NPC Routine Execution:** NPCs navigate the map autonomously according to believable daily routines. At a high level, each NPC goes through a full day cycle (morning, daytime, evening activities) and their locations/activities make sense (e.g. attending class in the morning, leisure or meals at noon, returning to dorm at night). The NPCs appear as **independent actors** living in the world rather than static quest-givers, enhancing the sense of a self-sufficient environment[[3]](https://game-design-snacks.fandom.com/wiki/NPC_Schedules_-_Help_Create_the_Illusion_of_a_Complex,_Self-Sufficient_World#:~:text=Older%20role,actors%20that%20the%20player%20happens). If the simulation runs through multiple in-game days, NPCs should repeat or adapt their schedule reliably each day.
* **Time-of-Day System:** A functional in-game clock or time mechanism exists and drives the schedules. This can be a simplified accelerated clock (for testing we might make 1 hour = 1 minute real-time, for example). The passage of time is visible or can be inferred (e.g. a clock UI or simply by NPC behavior changes). This ensures time-dependent events (like classes starting) happen correctly.
* **Placeholder Asset Integration:** All game content is represented by placeholder graphics or text labels, and these stand-ins are clearly documented. The game runs using these placeholders seamlessly – meaning that missing final art does not break the gameplay. For instance, every location on the map is visibly marked (even if just with a label or colored area) so testers know what it represents, and every NPC has a distinguishable placeholder sprite. The acceptance here is that one could play-test the game and understand what everything is supposed to be, despite the temporary art, and that swapping in final assets later will be straightforward.
* **Extensibility Demonstrated:** The design’s flexibility is proven by adding at least one new content element (such as a new NPC with a schedule or a new location on the map) during development and verifying the systems accommodate it. Successful integration of a new element without major refactoring is a key proof of extensibility. For example, if we add a “Library” area to the map and assign an NPC to go there in the afternoon, the engine should support this with minimal code changes. This criterion ensures our architecture is robust and future-proof.
* **Stability and No Regressions:** By the final milestone, the game should run for extended periods (e.g. multiple day cycles) without crashes or logic breakdowns. Earlier features (like player movement) should continue to work after later features (like NPC AI) are added. Each milestone’s features must remain functional in the final build. Essentially, passing all milestone tests means the overall system meets a basic quality bar for stability and functionality.

If all the above conditions are met, we will consider the project’s initial phase (engine with movement and scheduling) to be successfully completed. At that point, the foundation is laid to proceed into subsequent phases, which will likely involve replacing placeholder art with final graphics, adding interactive dialogue or missions, polishing the AI, and expanding the game world.

# Delegation Matrix – Milestones & Deliverables

Below is the **milestone-based breakdown** of the project, outlining each phase from start to finish (A to Z). Each milestone corresponds to a set of features or improvements that build upon the previous ones. Milestones are **ordered and gated** – we do not move to a later milestone until the prior one’s deliverables are completed and tested. Each milestone is defined by its **deliverable** (what will be achieved), key **tasks**, and **acceptance criteria** to verify completion. This breakdown serves as a strategic roadmap as well as a delegation plan, clarifying the phases of work and their sequence. (Note: Milestones are based on feature readiness, not tied to calendar dates – they are deliverable-focused and can be completed at whatever pace the team requires, though the numbering reflects the intended order.)

**Milestone 1: Placeholder Map & Basic Player Movement**  
**Deliverable:** A basic game world is set up with a placeholder map and a controllable player character. The player can walk around the environment using keyboard controls.  
**Tasks:** - Create a simple top-down **map layout** using placeholder graphics. For example, use colored rectangles or a rough floor-plan image to represent different rooms/areas (classrooms, hall, dormitory). Clearly label each area on the map (text or simple sign sprites) so testers know what they represent. Ensure there are colliders or boundaries for walls and impassable terrain. - Implement the **player character (PC) controls**: Allow the PC to move up, down, left, and right (or analog movement if supported) in the world. Use a placeholder sprite for the PC (e.g. a colored square or simple character icon). Animate movement with a basic sprite sheet if available, or at least show direction changes (this is optional polish at this stage; the core is the movement itself). - **Camera/Viewport** (if applicable): Set up the camera to follow the player if the map is larger than the screen. Alternatively, ensure the window shows enough of the map for testing movement. This may be trivial for a small map or if using an orthographic camera centered on the player. - **Collision & Boundaries:** Make sure the PC cannot walk through walls or outside the defined map area. This might involve adding collision detection to the placeholder map (e.g. blocking out walls) or simply coding boundary limits. - Test the movement in all areas of the map, adjust player speed if needed for a good feel, and confirm the map layout is fully navigable. Fix any stuck points or places the player can glitch through.

**Acceptance Criteria:** The player can freely move around the entire placeholder map without leaving the bounds or clipping through obstacles. All key locations on the map are reachable. Movement controls feel responsive (no significant lag or unregistered inputs). The placeholder map is visually clear in terms of layout (each region labeled). No game-breaking issues (e.g. crashes or getting irreversibly stuck) occur during exploration. Essentially, we have a *sandbox map* and a moving avatar, confirming the most basic interaction: navigation.

**Milestone 2: Introduce NPCs with Basic Schedules**  
**Deliverable:** One or more NPC characters are added to the game, each following a simple daily schedule on the map. An in-game clock is implemented to drive these NPC routines.  
**Tasks:** - Implement a **Time System**: Create an in-game clock or time counter. This could be as simple as an integer representing hours (e.g. 0–23 for hours in a day) that loops, or a more detailed day:hour:minute system. It can progress quickly for testing (for example, 1 game hour per real-time minute). Ensure we can reset or loop the day easily for repeated testing. - **NPC Entity Setup:** Create an NPC character class or object. Use placeholder sprites for NPCs (distinct from the player’s sprite, maybe different color to differentiate). Spawn a few NPC instances in appropriate starting locations (e.g. in their dorms or at a morning location). - **Define Schedules:** For each NPC, hardcode or data-drive a very basic schedule. For example, NPC1: at 8 (morning) go to Classroom A; at 12 (noon) go to Cafeteria; at 17 (5 PM) go to Dorm. NPC2: slightly different times or locations. Initially, keep schedules simple and maybe identical structure (all have 2-3 major movements per day). The schedule can be represented as a list of (time, target\_location) pairs. - **Movement AI:** When the clock hits a scheduled time, update the NPC’s state so that it “travels” to the target location. Implement pathfinding or simple movement for NPCs to navigate the map: this could be grid-based (if the map is tile-based under the hood) or node-based (predefine waypoints to each location) or even simplistic (NPC goes in a straight line if no obstacles, suitable only if map open – but better to handle obstacles). A *pathfinding on a grid of walkable tiles is a robust solution if feasible with placeholders. Otherwise, manually define routes for this milestone to keep it simple (e.g. NPC follows a specified path to get from dorm to class). - Ensure NPCs start moving at the correct times and reach their destinations. Possibly give them a constant walking speed. If they reach early, they can idle at the destination until next event. If they are late (if using a naive movement, they might not arrive by the next scheduled time), consider either speeding them up or allowing schedules to not overlap for now. - Make the NPC movement visible to the player. We might add a simple* *visual indicator of time*\* (like a UI clock or even printing time to console) for debugging, so we know what time it is in-game. - Synchronize multiple NPCs: if two NPCs have the same destination at the same time (e.g. two students to the same class), ensure they can both get there (maybe starting from different start points). They might clip through each other for now (that level of avoidance can be improved later), but at least they shouldn’t cause each other to jam or crash the game logic. - Test by letting the simulation run through a full day cycle. Observe that each NPC goes through their routine in the correct order. Adjust timing or paths if any NPC consistently fails to reach on time or at all.

**Acceptance Criteria:** The game now has active NPCs that the player can observe moving on their own. At least **two NPCs** (for example) have schedules where they go to different locations at designated times of day, and these movements happen correctly in-game. The in-game clock triggers events as expected (verified by seeing NPC movements at the right times). NPCs successfully navigate from their start points to their destinations (even if using rough paths). They do not walk through walls (unless intentionally allowed for a placeholder, which is not desired) and do not disappear off the map. The player can still move around freely and potentially follow any NPC throughout their day. The basic illusion of a **living world** is established: e.g. when the school “morning” starts, the player will see students (NPCs) walking to class on their own. This milestone is achieved when NPC daily movement is visible and repeatable without manual intervention.

**Milestone 3: NPC AI Refinement and Schedule Variability**  
**Deliverable:** A more robust and flexible NPC scheduling system with multiple NPCs, improved pathfinding/AI, and varied routines. NPC behaviors will be refined to appear less mechanical – introducing small variations or reactions to make them more lifelike.  
**Tasks:** - **Expand NPC Roster:** Add more NPCs (several students, maybe a teacher or two) to test the system’s scalability. Give each NPC a unique schedule or at least stagger some times/locations so not all NPCs move as a single unit. For example, NPC3 could have a different class or a free period when others are in class, etc. - **Improve Pathfinding:** If in Milestone 2 we used very simple movement, now is the time to implement a proper pathfinding mechanism (if not already). Ensure NPCs can find paths around obstacles and each other. This could involve using a grid-based A *algorithm for movement or Unity’s NavMesh or any path system appropriate to the engine. The goal is NPCs take sensible routes (e.g. through hallways, doors) to reach destinations instead of beelining through walls. -* *Avoidance & Collision Handling:* *Implement basic avoidance so NPCs don’t all overlap exactly. This might be as simple as a rule that if an NPC is very close to another, it slows down or chooses a slightly different path node to avoid clogging. We don’t need a perfect solution (crowd simulation can get complex), but at least NPCs shouldn’t indefinitely get stuck against each other. Also ensure NPCs yield or wait if their path is completely blocked (though with an open map or multiple routes this might not occur). -* *Schedule Variations:* *Introduce variability in schedules to make the world feel more dynamic. Options include: -* *Staggered times:* *e.g. NPCs start class at slightly different minutes (8:00, 8:05, 8:10) so they don’t move all at once. -* *Random events:* *e.g. There’s a small chance an NPC will deviate (skip going to cafeteria one day and go to library instead). This can be done via a simple random check in their schedule script. -* *Idle behaviors:* *When NPCs reach destinations early or have free time, give them an idle animation or wandering behavior. For instance, if a student arrives to class 10 minutes early, they might wander in the hallway or wait outside until class time. -* *Day/Night Cycle (if applicable):* *If we plan on a day-night visual indicator, implement a basic change (like background color shift or lighting change) as time passes. NPC schedules might include going to sleep at night (they could disappear inside dorms after a certain hour). This adds realism, though it’s largely cosmetic at this stage. -* *Robustness:* *Make sure adding or changing schedules is straightforward – for example, we should be able to easily adjust an NPC’s routine in data and see it in game. This likely involves refactoring schedule definitions into a cleaner format or scripting if we haven’t already. Ensure the system can handle edge cases (what if two events overlap for one NPC? Perhaps we log or assert to avoid that scenario for now). -* *Testing complex scenarios:*\* Run the simulation with all NPCs through several day cycles. Watch for any issues like NPCs getting permanently stuck, two NPCs colliding and freezing, or schedule events misfiring. Adjust logic to handle these (e.g. if NPC stuck, maybe teleport them to destination after a timeout as a failsafe, though in final game we’d avoid that). Also test performance: even with placeholders, ensure that, say, 10 NPCs moving doesn’t cause frame lag.

**Acceptance Criteria:** The NPC behavior now feels considerably richer and more resilient. We consider this milestone complete if:  
- We have a **small population of NPCs (e.g. 5–10)** each with their own routine, all coexisting smoothly on the map.  
- NPCs reliably navigate to their scheduled destinations on time *most* of the time, and any minor delays or detours still resolve naturally (they eventually get where they need to go).  
- The world simulation shows **variety**: not every NPC is doing the exact same thing at the exact same moment. From the player’s perspective, the school has different activities going on (some NPCs in class, some walking in halls, some at lunch, etc., depending on time). This fulfills the goal of a non-static world where NPCs appear to have independent lives.  
- Improvements like collision avoidance are evident: NPCs are not all clumping together or stuck. For example, if two NPCs head for the same doorway, they might naturally queue or find alternate routes rather than overlapping unrealistically.  
- The scheduling system proves flexible – during testing, we were able to tweak schedules or add a new NPC without any code breaking. (This is a **qualitative** criterion, but it’s important: it indicates our data-driven approach is working.)  
- No new crashes or major bugs have been introduced. The player can still move freely and nothing from Milestones 1 and 2 regressed. The game can run through multiple cycles (e.g. simulating 2-3 days in a row) without issues like memory leaks or logic errors accumulating.

At this stage, we should have a fully functional “sandbox” of the school where the player can wander and the NPCs live out their days. It may not be content-rich or pretty, but the engine underneath is doing the heavy lifting correctly.

**Milestone 4: Map Expansion and Extensibility Demonstration**  
**Deliverable:** An expansion of the game world (adding at least one new location/map section) and integration of that expansion into the existing systems to prove the engine’s extensibility. Additionally, preparation for upcoming features (like interactions) may be started by laying down some structural placeholders.  
**Tasks:** - **Add New Location:** Extend the current map or add a new scene representing a *new area*. For example, introduce a “Library” building or a “Gymnasium” that was not in the original layout. Use placeholders consistent with our style (perhaps another colored shape or an entirely new map image for a second scene). If using multiple scenes, implement a way for the player and NPCs to transition between scenes (e.g. walking through an exit that loads the new scene). If keeping one large map, simply append the new area to the existing map and put a connecting hallway. - **Incorporate into Schedules:** Update or add at least one NPC’s schedule to include the new location. For instance, have a student go to the Library at 3 PM as an extra activity in their day. Or, add a new NPC (e.g. a Librarian NPC) who mostly stays in the Library but perhaps goes to lunch at the cafeteria. This will test that our scheduling and movement system can handle a location that was not originally present. It will also test pathfinding across a larger map or multi-scene travel. - **Adjust Systems if Needed:** Expanding the map might reveal any hard-coded assumptions (like map boundaries or pathfinding grid size). Refactor any such parts to be data-driven. Ensure the time system and schedule manager can handle more events or longer travel distances (e.g. if it takes longer to walk to the library, does the NPC leave earlier or do we allow them to arrive late? Possibly adjust schedule times). - **Interaction Placeholders:** Since we know “we’ll add interactions later,” take this milestone as an opportunity to stub out how interactions *might* be added, to ensure nothing in our current engine design blocks it. For example: - Add a placeholder **“interact” key** for the player: pressing a certain key when near an NPC could simply pop up a dummy text like “(Placeholder) Hello!” or even just log a message. This doesn’t add full dialogue, but it shows that we can detect an interaction trigger (player in proximity and key pressed). - Mark NPCs or objects that would be interactive with some indicator (even just a different outline or an icon above their head) to ensure we can flag interactable entities in the future. - These steps ensure that when the time comes to implement actual interactions or cutscenes, our current architecture (movement, update loop, etc.) can accommodate pausing schedules or handling conversation mode. It’s easier to adjust now than retrofit later. - **Testing Expansion:** Thoroughly test that the new location works: the player can go there, NPCs can go there according to schedule, and transitions (if multi-scene) do not break anything. Also test the placeholder interaction to verify the concept (e.g. walk up to the new Librarian NPC and press the key, see the placeholder response). Monitor if any new bugs are introduced with the larger world, such as NPCs failing to find the new location or the player getting stuck at scene boundaries. - **Documentation Update:** Update any documentation (internal docs or code comments) to reflect the new additions. Especially, list all placeholder assets now in use (including those for the new area and any new NPC) so that the art team knows what to replace in the future. Note any new schedule or map configuration steps so the design team can add more locations/NPCs later easily.

**Acceptance Criteria:** This final milestone is achieved when we have convincingly shown that our engine can be extended and is ready for the next stages of development. Concretely:  
- A **new area is added and fully functional** in the game. The player can access it (walk into it or load it) and NPCs recognize it as part of their world (at least one NPC goes there as part of its routine). Everything that worked before still works with the bigger map (no NPC gets lost going to the new area, pathfinding accounts for it, etc.).  
- The introduction of new content did **not require major refactoring** of the core systems. (It’s fine if we had to extend a config file or add a new path node – but we shouldn’t have had to rewrite the movement engine to make this work.) This will typically be evaluated by reviewing the development effort: if adding one location or NPC is just a data addition and minor code tweaks, that’s a pass. If it broke a lot of code, then extensibility wasn’t truly achieved and needs revisiting.  
- The **placeholder interaction trigger** is in place and working in a basic form. For example, testers should be able to press the interact key near the designated NPC and see the placeholder feedback. This shows that the engine can support player-NPC interaction loops, which is crucial for the next phase when real interactions (like dialog or events) are implemented.  
- All previously implemented features still function after the expansion. The player movement, original NPC schedules, and time system should continue to operate as before. There should be no regression in functionality or new instabilities introduced by the larger world. Running a full day cycle with all NPCs (including those using the new area) should succeed without errors.  
- At this point, we effectively have a **vertical slice** of the simulation: a player can walk around a (placeholder) school, witness a number of NPCs going about their day, and perform a rudimentary interaction. The engine is content-complete for the prototype phase, meaning we can now safely proceed to add real content (art assets, dialogues, more story-driven behaviors) on this strong foundation.

With all milestones completed, the project has progressed from a simple walking demo to a functioning simulation prototype. Each milestone’s deliverable has been tested and meets the criteria defined, ensuring that our development is on solid footing. **The next steps beyond this plan** would involve transforming this barebones simulation into a polished game: replacing placeholders with final pixel art, deepening the interaction systems (so the player can talk to or influence NPCs), adding game objectives or storylines, and iterating on any gameplay feedback. Thanks to the groundwork laid out in these milestones, the team can undertake those tasks with confidence that the core engine (movement, AI scheduling, extensible world structure) is reliable and well-documented. This phased, deliverable-driven approach also means we always have a working build to show stakeholders or use for playtesting at each stage, mitigating risk and aligning development with the project’s strategic goals.

[[1]](https://playgama.com/blog/general/how-can-i-design-npc-schedule-mechanics-like-those-in-stardew-valley-for-caroline-to-enhance-player-interaction-in-my-game/#:~:text=NPC%20schedule%20mechanics%20are%20crucial,similar%20to%20Stardew%20Valley%E2%80%99s%20Caroline) How can I design NPC schedule mechanics, like those in Stardew Valley for Caroline, to enhance player interaction in my game? - Playgama Blog

<https://playgama.com/blog/general/how-can-i-design-npc-schedule-mechanics-like-those-in-stardew-valley-for-caroline-to-enhance-player-interaction-in-my-game/>

[[2]](https://blog.lipsumhub.com/placeholder-content-in-game-development/#:~:text=The%20use%20of%20placeholder%20,in%20creating%20the%20final%20assets) Placeholder Content in Game Development - Lipsum Hub

<https://blog.lipsumhub.com/placeholder-content-in-game-development/>

[[3]](https://game-design-snacks.fandom.com/wiki/NPC_Schedules_-_Help_Create_the_Illusion_of_a_Complex,_Self-Sufficient_World#:~:text=Older%20role,actors%20that%20the%20player%20happens) NPC Schedules - Help Create the Illusion of a Complex, Self-Sufficient World | Game Design Snacks Wikia | Fandom

<https://game-design-snacks.fandom.com/wiki/NPC_Schedules_-_Help_Create_the_Illusion_of_a_Complex,_Self-Sufficient_World>