Three Game Mechanics  
*Learning Outcome 1 (some amount of 15%?)*

## 1. In-Game Timer

**What did you implement?**A modular in-game timer that smoothly interpolates the world clock from a specified start time (e.g. 00:00) to an end time (e.g. 18:00) over a configurable real-time duration (e.g. 10 minutes). It exposes getters and setters for start/end times and duration.

**What did you research?**

* Godot’s Timer node and its signals (see Godot docs: Timer)
* Articles on writing reusable components in GDScript (e.g. GDQuest’s “Building Reusable Code”)

**What did you try? What worked? What did not?**

* Tried: Hard-coding time values with repeated if/match statements.  
  Outcome: Worked, but was verbose and brittle.
* Refactored to: A single Timer node driving a progress = elapsed / duration value, then interpolating time with lerp(start, end, progress).  
  Outcome: Flexible, concise and easy to tweak via exported variables.

**What did you learn?**

* How to decouple logic from data by using exported properties and signals.
* The power of Godot’s built-in nodes to eliminate boilerplate code.

**How can you apply what you learned to future games?**Any mechanic that evolves over time, day/night cycles, timed power-ups, animation blending—can be implemented as a self-contained, parameter-driven scene.

**What did you find most challenging professionally? How did you overcome it?**Refactoring without introducing regressions was tricky. I overcame this by:

1. Writing quick play-mode tests for edge cases (zero duration, start > end).
2. Pair-programming with a teammate to review each refactoring step.

## 2. Animations & Game-Map Creation

**What did you implement?**A full 3D environment map modelled in Blender, plus rigging and skinning of a humanoid character for export and animation in Godot.

**What did you research?**

* Blender fundamentals (official manual: Modelling & Rigging chapters)
* Tutorials on exporting animated rigs to Godot (e.g. “Godot–Blender Character Pipeline”)

**What did you try? What worked? What did not?**

* Tried: Writing a custom GLSL shader in C# to layer map segments.  
  Outcome: Failed, object layering was mis-ordered and shader didn’t align.
* Switched to: Applying materials and modular mesh placement directly in Godot.  
  Outcome: Reliable visual results and simpler workflow.

**What did you learn?**

* The Blender→Godot pipeline nuances (bone naming, scale issues).
* When to abandon over-engineering (custom shaders) in favour of built-in tools.

**How can you apply what you learned to future games?**I’ll define clear export settings and naming conventions up-front, and prototype visual ideas in the engine before investing time in custom shaders.

**What did you find most challenging professionally? How did you overcome it?**Learning Blender from scratch under time pressure was daunting. I overcame it by:

1. Breaking the task into small goals (room blocking, mesh cleanup, basic rig).
2. Following step-by-step video tutorials to build confidence.

## 3. From Teleporting Monster → Moving Monster

**What did you implement?**Enhanced the enemy AI so that, instead of teleporting instantly to the player’s room, the monster now follows a navigable path and faces the correct direction while walking.

**What did you research?**

* Godot NavigationServer and pathfinding basics (Godot docs: Navigation)
* Vector math for smooth rotation (e.g. angle\_to\_point, lerp\_angle)

**What did you try? What worked? What did not?**

* Tried: Manually updating global\_position each frame without delta scaling.  
  Outcome: Movement speed varied wildly on different equipment.
* Fixed by: Multiplying by delta and using move\_and\_slide() for frame-rate-independent motion.
* Tried: Hard-coding direction checks.  
  Outcome: Clunky turning behaviour.
* Improved with: lerp\_angle(current\_angle, target\_angle, turn\_speed \* delta) for smooth rotation.

**What did you learn?**

* Always factor in delta for consistent movement.
* How to combine path arrays with angle interpolation for natural enemy motion.

**How can you apply what you learned to future games?**Any character or object requiring movement can use the same pattern: compute a path, feed waypoints into a delta-scaled motion and rotation routine.

**What did you find most challenging professionally? How did you overcome it?  
Integrating new pathing logic into an unfamiliar codebase was difficult. I overcame it by:**

1. Commenting out old teleport code and adding small chunks of new logic incrementally.
2. Visualising the path in debug mode (drawing lines between waypoints) to confirm correctness before polishing.