Requirement’s specification of Oregon Trail Survival

|  |  |
| --- | --- |
| Client | Faculty of Engineering, Design and Applied Sciences – Barberi |
| User | Students and instructors involved in the “Oregon Trail Survival” educational game. Players act as pioneers traveling from Independence, Missouri, to the Willamette Valley in Oregon in 1848. |
| Funcional requirements | *FR1 – Movement and exploration FR2 – Scenario definition and transitions FR3 – Map boundaries and collision detection FR4 – Combat system with firearms FR5 – Survival and resource management FR6 – Random enemy generation FR7 – Achievement tree with binary search tree FR8 – User interface with animations and indicators FR9 – Dynamic dialogues with Gemini API FR10 – Aiming crosshair  FR11 – Health rule: three hits before death FR12 – Limited ammunition management FR13 – Resource collection FR14 – Main menu screen FR15 – Scenario screens with indicators FR16 – Game Over screen FR17 – Victory screen FR18 – Achievements screen (BST viewer) FR19 – Concurrent animations using threads* |
| Problem context | This project is a modern, survival-style adaptation of the classic educational game The Oregon Trail. The system simulates the historical journey with integrated action and resource-management mechanics. Players must move through three connected scenarios (Plains, Rocky Mountains and Columbia River), facing hostile automatons, managing food, ammunition and medical supplies, and making strategic decisions to survive. The application will be developed in Java with JavaFX for the interface and will include features such as concurrency (animations), binary search trees for achievements, and linked lists for resource management. |
| Non-functional requirements | *NFR1 – The whole game must be built in Java with JavaFX, without using external game engines. NFR2 – It should run smoothly and be responsive on normal lab computers. NFR3 – The interface must be clear and easy to use, always showing health, inventory, ammo, and achievements. NFR4 – The code must be maintainable and follow good coding and design practices. NFR5 – There must be an in-app user manual shown in a JavaFX pop-up, explaining controls, indicators, and basic strategies. NFR6 – It’s not allowed to use libraries like OpenGL or any other external game engine. NFR7 – If generative AI is used, it has to be justified, cited, and integrated in a critical way.* |
| Process requirements | *PR1 – Development must follow Test-Driven Development (TDD) with unit tests in JUnit. PR2 – The project must use Git/GitHub with at least 15 commits, and each commit history should include quality metrics like error density, reliability, and completeness. PR3 – Work must be fairly distributed among all team members, and contribution equality must be verifiable through the commits. PR4 – Deliverables must be submitted on schedule:*   * *Week 10: requirements, test designs, first test code, and class diagrams.* * *Week 14: updated docs and design, linked lists and binary trees, sorting, binary search, and logic for the model.* * *Week 18: final delivery and defense, with updated design, JavaFX interface, animations (using threads), full integration of the model with the view, and final documentation. PR5 – The project must apply the good practices and design patterns taught in class.* |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identifier and name | *FR1 – Movement and exploration* | | | |
| Summary | The system must allow the player to move in four directions (up, down, left, right) within the map, ensuring collisions with walls or map boundaries are not allowed. | | | |
| Inputs | **Input name** | **Data type** | | **Condition valid values** |
| direction | String | | Must be one of: "up", "down", "left", "right" |
| currentPosition | Integer pair (x,y) | | Must be within current map limits |
| Result or postcondition | The player’s position is updated and remains within valid map boundaries. | | | |
| Outputs | **Output name** | | **Data type** | **Format** |
| newPosition | | Integer pair (x,y) | (x,y) coordinates on the current map |
| updatedScreen | | Graphic | Visual representation of the new player location |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identifier and name | *FR2 – Scenario definition and transitions* | | | |
| Summary | The system must define three main scenarios (Plains, Rocky Mountains, and Columbia River) and allow the player to move between them through specific transition points such as doors, river crossings, or mountain passes, loading the new map while maintaining player state. | | | |
| Inputs | **Input name** | **Data type** | | **Condition valid values** |
| currentPosition | Integer pair (x,y) | | *Must match the coordinates of a defined transition point* |
| reachTransitionPoint | Boolean | | Must be true when the player reaches a valid transition point |
| Result or postcondition | The current scenario is updated to the new one, and the player’s position and state are transferred to the correct starting point of the next map. | | | |
| Outputs | **Output name** | | **Data type** | **Format** |
| newScenario | | String | Name of the loaded scenario: "Plains", "RockyMountains", or "ColumbiaRiver" |
| updatedPosition | | Integer pair (x,y) | Must be true when the player reaches a valid transition point |
| screenRefresh | | Graphic | Visual representation of the new map with the player in the correct location |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identifier and name | ***FR3 – Map boundaries and collision detection*** | | | |
| Summary | The system must define map borders and impassable obstacles for every scenario. Collision detection must prevent the player from moving outside the defined playable area or crossing solid objects such as walls, rocks or rivers, and it must update in real time as the player moves. | | | |
| Inputs | **Input name** | **Data type** | | **Condition valid values** |
| currentPosition | Integer pair (x, y) | | Must represent a valid coordinate inside the current map |
| desiredPosition | Integer pair (x, y) | | Must be calculated based on the intended movement direction |
| Result or postcondition | If the desired position is inside the map and not blocked, the player’s position is updated. If the desired position is outside the map or collides with an obstacle, movement is blocked and the player’s position remains unchanged. | | | |
| Outputs | **Output name** | | **Data type** | **Format** |
| updatedPosition | | Integer pair (x, y) | (x, y) coordinates of the valid final position |
| collisionStatus | | Boolean | True if movement is blocked by a border or obstacle |
| visualFeedback | | Graphic | Optional effect or message indicating a collision (e.g., shake or sound) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identifier and name | *FR4 – Combat system with firearms* | | | |
| Summary | The system must implement a combat system that allows the player to attack with two types of firearms (a muzzle-loading rifle and a revolver) each with different damage and reload time. The system must decrease ammunition after every shot, apply the correct damage to enemies, and handle reload actions when ammunition is exhausted | | | |
| Inputs | **Input name** | **Data type** | | **Condition valid values** |
| weaponSelected | String | | Must be one of: "rifle", "revolver" |
| fireCommand | Boolean | | Must be true when the player clicks or presses the shoot control |
| reloadCommand | Boolean | | Must be true when the player presses the reload control |
| targetCoordinates | Integer pair (x, y) | | Must be within current map boundaries |
| Result or postcondition | Enemy health decreases according to the weapon’s damage. The player’s ammunition count is updated. If ammunition reaches zero, a reload action is required before firing again. | | | |
| Outputs | **Output name** | | **Data type** | **Format** |
| updatedAmmo | | Integer | Remaining bullets shown as a number |
| enemyHealth | | Integer | Remaining health points of the targeted enemy |
| shotAnimation | | Graphic | Visual effect of firing and impact |
| reloadStatus | | Boolean | True if weapon is reloaded and ready to fire |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identifier and name | ***FR5 – Survival and resource management*** | | | |
| Summary | The system must track the player’s health and manage a limited inventory of essential resources such as food, medicine, and ammunition. Players can collect resources from the environment, use them to recover health or perform actions, and discard them if needed. All stored data must be implemented using linked lists. | | | |
| Inputs | **Input name** | **Data type** | | **Condition valid values** |
| collectedItem | String | | Must be one of: "food", "medicine", "ammunition" |
| actionType | String | | Must be one of: "collect", "use", "discard" |
| itemQuantity | Integer | | Must be greater than 0 |
| healthImpact | Integer | | Optional; applied when using food or medicine |
| Result or postcondition | The player’s inventory and health indicators are updated. If resources are used, their quantity decreases accordingly. If health reaches zero, a game-over condition is triggered. | | | |
| Outputs | **Output name** | | **Data type** | **Format** |
| updatedInventory | | Linked list of items | Each element stores item name and remaining quantity |
| updatedHealth | | Integer | Value between 0 and maximum health |
| inventoryDisplay | | Graphic | Visual representation of the current inventory and health on the screen |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identifier and name | ***FR6 – Random enemy generation*** | | | |
| Summary | The system must randomly generate hostile enemies with simple artificial intelligence (AI) that pursue and attack the player. Enemy creation must vary in number, timing, and location to make every gameplay session unique and unpredictable. | | | |
| Inputs | **Input name** | **Data type** | | **Condition valid values** |
| generationEvent | Event / Timer | | Occurs periodically or when predefined game conditions are met |
| mapArea | Integer pair (x, y) range | | Must represent valid coordinates inside the current scenario |
| difficultyLevel | Integer | | Must be within predefined difficulty settings |
| Result or postcondition | A set of enemies is spawned on the current map with random positions and basic AI that moves toward the player. Each enemy remains active until defeated or until the scenario ends. | | | |
| Outputs | **Output name** | | **Data type** | **Format** |
| spawnedEnemies | | Linked list of enemy objects | Each element includes position, health, and state |
| enemyPositions | | List of integer pairs (x, y) | Coordinates of all active enemies |
| spawnNotification | | Graphic / Sound | Visual or audio cue indicating new enemies have appeared |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identifier and name | ***FR7 – Achievement tree with binary search tree*** | | | |
| Summary | The system must record and display player achievements using a binary search tree. Each achievement is stored as a node ordered by a chosen key (for example, achievement name or completion time). Players must be able to open a dedicated achievements window to view all unlocked achievements in an organized tree format. | | | |
| Inputs | **Input name** | **Data type** | | **Condition valid values** |
| achievementName | String | | Must match the name of an existing achievement |
| completionTime | Date/Time | | Must represent the moment when the achievement is earned |
| achievementData | Object | | Must contain description and reward information |
| Result or postcondition | The binary search tree is updated with the new achievement node. The achievements window displays all unlocked achievements in the correct order. | | | |
| Outputs | **Output name** | | **Data type** | **Format** |
| UpdatedAchievementTree | | Binary search tree | Nodes ordered by the chosen key |
| achievementDisplay | | Graphic | Tree view showing each unlocked achievement and its details |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identifier and name | ***FR8 – User interface with animations and indicators*** | | | |
| Summary | The system must implement a graphical user interface (GUI) that displays all essential game information such as player health, inventory, ammunition, and current scenario and includes animated visual elements to enhance player experience. The interface must be clear, responsive, and updated in real time during gameplay. | | | |
| Inputs | **Input name** | **Data type** | | **Condition valid values** |
| playerData | Object | | Must include current health, inventory content, and ammunition count |
| scenarioData | Object | | Must describe the current scenario and relevant indicators |
| animationEvent | Event | | Must correspond to valid in-game actions such as movement, attack, or scenario change |
| Result or postcondition | All interface components (indicators, panels, animations) are refreshed to reflect the latest game state without delay. | | | |
| Outputs | **Output name** | | **Data type** | **Format** |
| interfaceDisplay | | Graphic | Updated GUI showing health, inventory, ammunition, and scenario |
| animationEffect | | Graphic / Sound | Visual or audio feedback corresponding to the latest action |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identifier and name | ***FR9 – Dynamic dialogues with Gemini API*** | | | |
| Summary | The system must integrate the Gemini API to generate dynamic dialogues for enemies or non-player characters (NPCs) during different in-game situations, such as after being defeated or when interacting peacefully. The generated text must adapt to the specific context of each interaction and appear instantly on the game screen. | | | |
| Inputs | **Input name** | **Data type** | | **Condition valid values** |
| interactionType | String | | Must be one of: "enemy\_defeated", "enemy\_encounter", "npc\_conversation" |
| scenarioContext | Object | | Must include current scenario name and key event detail |
| playerStatus | Object | | Must include relevant information such as player health or inventory status |
| Result or postcondition | The Gemini API provides a context-appropriate dialogue, which is displayed to the player and stored in the current game session. | | | |
| Outputs | **Output name** | | **Data type** | **Format** |
| generatedDialogue | | String | Text generated by the Gemini API |
| dialogueDisplay | | Graphic | Visual text box or speech bubble shown on screen |
| dialogueLog | | File or memory object | Optional record of dialogues for the session |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identifier and name | *FR10 – Aiming crosshair* | | | |
| Summary | The game shows a crosshair that follows the mouse; combat uses it to aim shots. | | | |
| Inputs | **Input name** | **Data type** | | **Condition valid values** |
| mousePos | Integer pair (x,y) | | Inside the viewport |
| isAiming | Boolean | | *True* when aiming is active |
| Result or postcondition | Crosshair position updates smoothly and is available to the shooting logic. | | | |
| Outputs | **Output name** | | **Data type** | **Format** |
| crosshair | | Graphic | Crosshair drawn at (x,y) |
| aimHint | | Graphic | Optional HUD hint while aiming |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identifier and name | *FR11 – Health rule: three hits before death* | | | |
| Summary | The player has three hits (hearts). When hearts reach 0, it’s Game Over. | | | |
| Inputs | **Input name** | **Data type** | | **Condition valid values** |
| damageTaken | Integer | | ≥ 0 |
| healAmount | Integer | | ≥ 0 |
| Result or postcondition | Hearts update within [0..3]. At 0, trigger Game Over state. | | | |
| Outputs | **Output name** | | **Data type** | **Format** |
| heartsUI | | Integer | 0–3 hearts/icons |
| deathFlag | | Boolean | True when hearts == 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identifier and name | *FR12 – Limited ammunition management* | | | |
| Summary | Ammo is tracked per weapon and is limited; you can’t shoot if ammo is zero. | | | |
| Inputs | **Input name** | **Data type** | | **Condition valid values** |
| weaponSelected | String | | "rifle" / "revolver" |
| ammoChange | Integer | | Positive (pickup) or negative (shot) |
| Result or postcondition | Ammo count updates; shooting blocked at 0 until reloaded. | | | |
| Outputs | **Output name** | | **Data type** | **Format** |
| ammoDisplay | | Integer | Current ammo for selected weapon |
| shootBlocked | | Boolean | True when ammo == 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identifier and name | *FR13 – Resource collection* | | | |
| Summary | The player can collect food, medicine, and ammunition from the world. | | | |
| Inputs | **Input name** | **Data type** | | **Condition valid values** |
| pickup | Item | | food / medicine / ammunition |
| playerPos | Integer pair (x,y) | | Matches pickup location |
| Result or postcondition | Pickup disappears from the map and is added to the inventory. | | | |
| Outputs | **Output name** | | **Data type** | **Format** |
| updatedInventory | | Linked list | Items with quantities |
| pickupFX | | Graphic/Sound | Small feedback effect |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identifier and name | *FR14 – Main menu screen* | | | |
| Summary | A main menu lets you start the game or view achievements. | | | |
| Inputs | **Input name** | **Data type** | | **Condition valid values** |
| menuChoice | String | | "start", "achievements", "exit" (as designed) |
| Result or postcondition | Chosen option opens the correct screen or exits. | | | |
| Outputs | **Output name** | | **Data type** | **Format** |
| menuScreen | | Graphic | Rendered main menu |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identifier and name | *FR15 – Scenario screens with indicators* | | | |
| Summary | Scenario screens show the map plus HUD indicators (health, inventory, ammo). | | | |
| Inputs | **Input name** | **Data type** | | **Condition valid values** |
| playerData | Object | | Health, inventory, ammo present |
| scenarioId | String | | Current scenario identifier |
| Result or postcondition | HUD updates in real time to reflect the latest state. | | | |
| Outputs | **Output name** | | **Data type** | **Format** |
| scenarioView | | Graphic | Map + HUD indicators |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identifier and name | *FR16 – Game Over screen* | | | |
| Summary | When the player dies, show the Game Over screen. | | | |
| Inputs | **Input name** | **Data type** | | **Condition valid values** |
| deathFlag | Boolean | | *True* when hearts == 0 |
| Result or postcondition | Gameplay stops; options like restart or main menu are offered. | | | |
| Outputs | **Output name** | | **Data type** | **Format** |
| gameOverScreen | | Graphic | Game Over UI |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identifier and name | *FR17 – Victory screen* | | | |
| Summary | When the journey is completed, show the Victory screen. | | | |
| Inputs | **Input name** | **Data type** | | **Condition valid values** |
| victoryEvent | Event | | Triggered by final objective completion |
| Result or postcondition | Victory UI appears; allow returning to menu or viewing achievements. | | | |
| Outputs | **Output name** | | **Data type** | **Format** |
| victoryScreen | | Graphic | Victory UI |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identifier and name | *FR18 – Achievements screen* | | | |
| Summary | Unlocked achievements are stored in a Binary Search Tree and displayed in a dedicated window. | | | |
| Inputs | **Input name** | **Data type** | | **Condition valid values** |
| achievementEvent | Object | | Data for a newly earned achievement |
| Result or postcondition | BST updates with the new node; achievements are shown in order. | | | |
| Outputs | **Output name** | | **Data type** | **Format** |
| achievementsWindow | | Graphic | Tree/list view of achievements |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identifier and name | *FR19 – Concurrent animations using threads* | | | |
| Summary | Animations (movement, attacks, effects) run concurrently so the game feels real-time. | | | |
| Inputs | **Input name** | **Data type** | | **Condition valid values** |
| tick | Event/Timer | | Fired by the game loop/UI thread |
| Result or postcondition | Animations update independently without freezing gameplay. | | | |
| Outputs | **Output name** | | **Data type** | **Format** |
| animationFrames | | Graphic | Smooth animated visuals |

Second part of requirements specification: Dorfman requirements

**Step 1. Classification of Requirements (R1–R19)**

* R1 Movement and exploration
* R2 Scenario definition and transitions
* R3 Map boundaries and collision detection
* R4 Combat system with firearms (rifle & revolver)
* R5 Survival and resource management (health, inventory, resources)
* R6 Random enemy generation
* R7 Achievement tree with a Binary Search Tree (BST)
* R8 User interface with animations and indicators
* R9 Dynamic dialogues with Gemini API
* R10 Aiming crosshair (mouse)
* R11 Health rule: three hits before death
* R12 Limited ammunition management
* R13 Resource collection (food, medicine, ammunition)
* R14 Main menu screen
* R15 Scenario screens with HUD indicators (health, inventory, ammo)
* R16 Game Over screen
* R17 Victory screen
* R18 Achievements screen (BST viewer)
* R19 Concurrent animations using threads

**Step 2. Identification of Entities**

* Player: moves, shoots, manages inventory and health (3‑hit rule), interacts with world.
* Enemy: randomly spawned, pursues and attacks; has position/HP/state.
* Scenario: Plains, Rocky Mountains, Columbia River; holds map, spawns, transitions.
* MapGrid & TransitionPoint: spatial layout, bounds/collision, gates between scenarios.
* Item (Food/Medicine/Ammunition): collectible resources stored in inventory.
* Weapon (Rifle/Revolver): chosen to attack; ammo and reload behavior.
* AchievementTree / AchievementNode (BST): stores unlocked achievements in order.
* UI (JavaFX): menu, scenario/HUD, end states, achievements view; animated indicators.
* Dialogue (Gemini): provides context-aware lines; fallback if API fails.
* Animation/Loop (Threads): drives concurrent animations and keeps UI responsive.

**Step 3. Subsystem Hierarchy**

* Subsystem A – Exploration & Scenarios: movement, transitions, collisions, scenario‑HUD binding.
* Subsystem B – Combat & Enemies: weapons, aiming, ammo, enemy spawn/AI.
* Subsystem C – Survival & Resources: health rule, inventory, pickups.
* Subsystem D – Achievements (BST): record and visualize achievements.
* Subsystem E – Interface & Dialogues: menu, scenario screens, end states, Gemini.
* Subsystem F – Concurrency & Animation: threads/game loop and async UI updates.

**Step 4. First Assignment (Requirements → Subsystems)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Requirement** | **A** | **B** | **C** | **D** | **E** | **F** |
| R1 | ● |  |  |  |  |  |
| R2 | ● |  |  |  |  |  |
| R3 | ● |  |  |  |  |  |
| R4 |  | ● |  |  |  |  |
| R5 |  |  | ● |  |  |  |
| R6 |  | ● |  |  |  |  |
| R7 |  |  |  | ● |  |  |
| R8 |  |  |  |  | ● |  |
| R9 |  |  |  |  | ● |  |
| R10 |  | ● |  |  |  |  |
| R11 |  |  | ● |  |  |  |
| R12 |  | ● |  |  |  |  |
| R13 |  |  | ● |  |  |  |
| R14 |  |  |  |  | ● |  |
| R15 | ● |  |  |  | ● |  |
| R16 |  |  |  |  | ● |  |
| R17 |  |  |  |  | ● |  |
| R18 |  |  |  | ● | ● |  |
| R19 |  |  |  |  |  | ● |

**Step 5. Sub‑specification of Requirements (by Program)**

1. Program A – Exploration & Scenarios (R1, R2, R3, R15)

* R1\_A – Movement: compute desired tile from input; allow only orthogonal moves; check bounds/collision; update player position.
* R2\_A – Transitions: detect gate at player position; load target scenario; place player at entry point; preserve player state.
* R3\_A – Collision: maintain map obstacles/borders; block invalid moves; provide minimal feedback when blocked.
* R15\_A – Scenario HUD: bind scenario/player state to HUD indicators; refresh in real time.

1. Program B – Combat & Enemies (R4, R6, R10, R12)

* R4\_A – Combat: choose weapon; on fire, reduce ammo and apply damage; enforce reload when empty.
* R6\_A – Spawning & AI: spawn enemies per scenario rules; enemies pursue player and attempt attacks when in range.
* R10\_A – Aiming: track mouse crosshair; expose aiming vector/coords to combat.
* R12\_A – Ammo: track per‑weapon ammo; block firing at zero; update HUD.

1. Program C – Survival & Resources (R5, R11, R13)

* R5\_A – Inventory/Health: manage list of items; support collect/use/discard; reflect changes on HUD.
* R11\_A – Three‑hit rule: clamp hearts in [0..3]; when 0, raise Game Over event/state.
* R13\_A – Pickups: when colliding with a pickup, remove from map and add to inventory; play a simple FX.

1. Program D – Achievements (BST) (R7, R18)

* R7\_A – Record: insert achievement nodes ordered by key; support search.
* R18\_A – View: traverse BST to build a read‑only view; display in a separate window/screen.

1. Program E – Interface & Dialogues (R8, R9, R14, R16, R17)

* R8\_A – GUI: present HUD and animated feedback; keep indicators in sync with game state.
* R9\_A – Dialogues: request context‑aware lines from Gemini; on failure, use fallback text; show bubble/box and optionally log.
* R14\_A – Menu: display main menu; route to start game or achievements.
* R16\_A – Game Over: show end‑state view when death is triggered; stop gameplay; offer restart/menu.
* R17\_A – Victory: show end‑state view on completion; allow returning to menu/achievements.

1. Program F – Concurrency & Animation (R19)

* R19\_A – Loop/Threads: drive frame/timer events; update animations independently; avoid UI freezes (proper JavaFX threading).

**Step 6. Second‑Level (Modules)**

* A1 Movement Controller: input → valid moves.
* A2 Transition Manager: gate detection → scenario switch.
* A3 Collision Service: obstacle/bounds queries.
* A4 Scenario‑HUD Binder: expose scenario/player state to HUD.
* B1 Weapon Controller: shoot/reload/ammo sync.
* B2 Aiming Module: mouse crosshair → aim vector.
* B3 Enemy Spawner: timed/conditioned spawns.
* B4 Enemy AI: pursue/attack behavior.
* C1 Inventory Manager: linked‑list item storage; actions.
* C2 Health Manager: enforce three‑hit rule and UI flags.
* D1 Achievement BST: insert/search.
* D2 Achievements View: read‑only renderer.
* E1 Menu View: main navigation.
* E2 Scenario View & HUD: map + indicators; animation hooks.
* E3 End‑State Views: Game Over / Victory.
* E4 Dialogue Manager: Gemini integration + fallback.
* F1 Game Loop / Threads: timers, frame scheduling.
* F2 Async UI Updates: JavaFX thread safety.

**Step 7. Second Assignment (Programs ↔ Requirements)**

* Program A: R1, R2, R3, R15
* Program B: R4, R6, R10, R12
* Program C: R5, R11, R13
* Program D: R7, R18
* Program E: R8, R9, R14, R16, R17
* Program F: R19