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*** |
| 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 | * The system must be implemented entirely in Java with JavaFX, without external game engines. * It must support smooth performance and responsive controls on standard lab computers. * It should provide clear, intuitive screens and indicators for health, inventory and achievements. * It must be maintainable and well-documented, following good coding and design practices. |
| Process requirements | * The project will follow Test-Driven Development (TDD) with unit tests created in JUnit. * Version control will be managed with Git and GitHub, including at least 15 commits reporting quality metrics (error density, reliability, completeness). * Work will be divided and tracked to ensure equitable contributions by all group members. * Deliverables will be completed in phases: week 10 (requirements, test design, class diagrams), week 14 (data structures and logic), and final presentation in week 18. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 |

Requirements Specification – Dorfman Method

**Step 1. Classification of Requirements**

* R1: Movement and exploration in four directions.
* R2: Definition and transition of scenarios (Plains, Rocky Mountains, Columbia River).
* R3: Map boundaries and collision detection.
* R4: Combat system with firearms (rifle and revolver).
* R5: Survival and resource management (health, food, medicine, ammunition).
* R6: Random enemy generation.
* R7: Achievement tree implemented with a binary search tree.
* R8: User interface with animations and indicators.
* R9: Dynamic dialogues with Gemini API.

**Step 2. Identification of Entities**

* Entity 1: Player → controls character, manages inventory, combat, and survival.
* Entity 2: Enemies → generated with AI to pursue and attack the player.
* Entity 3: Scenarios → Plains, Rocky Mountains, Columbia River.
* Entity 4: Resources → food, medicine, ammunition.
* Entity 5: Weapons → rifle and revolver with different mechanics.
* Entity 6: Achievement tree → binary search tree of rewards.
* Entity 7: Gemini API → external system for dynamic dialogues.
* Entity 8: User interface (JavaFX) → screens, indicators, animations.
* Entity 9: Testing and version control system (JUnit + Git) → ensures software quality.

**Step 3. Subsystem Hierarchy**

* Subsystem A: Exploration and Scenarios (R1, R2, R3).
* Subsystem B: Combat and Enemies (R4, R6).
* Subsystem C: Survival and Resources (R5).
* Subsystem D: Achievements (R7).
* Subsystem E: Interface and Dialogues (R8, R9).

**Step 4. First Assignment of Requirements to Subsystems**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Requirement** | **Sub. A** | **Sub. B** | **Sub. C** | **Sub. D** | **Sub. E** |
| R1 | X |  |  |  |  |
| R2 | X |  |  |  |  |
| R3 | X |  |  |  |  |
| R4 |  | X |  |  |  |
| R5 |  |  | X |  |  |
| R6 |  | X |  |  |  |
| R7 |  |  |  | X |  |
| R8 |  |  |  |  | X |
| R9 |  |  |  |  | X |

**Step 5. Sub-specification of Requirements**

**Program A: Exploration and Scenarios**

* R1\_A: Player movement in four directions.
* R2\_A: Scenario transitions between Plains, Rocky Mountains, and Columbia River.
* R3\_A: Collision detection and map boundaries.

**Program B: Combat and Enemies**

* R4\_A: Firearms combat system (rifle and revolver).
* R6\_A: Random generation of enemies with AI.

**Program C: Survival and Resources**

* R5\_A: Health system and resource management (food, medicine, ammunition).

**Program D: Achievements**

* R7\_A: Recording achievements in a binary search tree.
* R7\_B: Visualization of achievements in a tree window.

**Program E: Interface and Dialogues**

* R8\_A: Graphical user interface with animations and indicators.
* R9\_A: Dynamic dialogues generated by Gemini API.

**Step 6. Second-Level Partitioning**

**Subsystem A (Exploration)**

* Module A1: Map movement.
* Module A2: Scenario transitions.
* Module A3: Collision detection.

**Subsystem B (Combat and Enemies)**

* Module B1: Firearms management.
* Module B2: Enemy generation and behavior.

**Subsystem C (Survival)**

* Module C1: Inventory system.
* Module C2: Health management.

**Subsystem D (Achievements)**

* Module D1: Achievement recording.
* Module D2: Achievement visualization.

**Subsystem E (Interface and Dialogues)**

* Module E1: Main screens and indicators.
* Module E2: Animated interface elements.
* Module E3: Gemini API dialogues.

**Step 7. Second Assignment**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Requirement** | **Program A** | **Program B** | **Program C** | **Program D** | **Program E** |
| R1 | X |  |  |  |  |
| R2 | X |  |  |  |  |
| R3 | X |  |  |  |  |
| R4 |  | X |  |  |  |
| R5 |  |  | X |  |  |
| R6 |  | X |  |  |  |
| R7 |  |  |  | X |  |
| R8 |  |  |  |  | X |
| R9 |  |  |  |  | X |