Oregon Trail

- Section 1.1 – Problem Statement & Objectives
    - *Domain*: Media and Entertainment
    - *Problem*: The kids need a fun game to play involving clips software.
    - *Objectives*: Create an enjoyable experience for people of all ages while educating people about the Oregon Trail and its historical relevance.
    - *Scope Boundaries*: The game does not have any form of medium other than text. This game is also limited in its depth and thus historical accuracy.
- Section 1.2 – Knowledge Representation & Ontology Scope
    - *Representation*: Entity Oriented Approach with rules governing how they interact with other entities.
    - *Core Entities*: Player, Wagon, Companion NPCs, Neutral NPCs, Hostile NPCs, Wagon Inventory, Environment, Forts
    - *Relationships*:
        - canTrade(Player, Neutral NPCs),
        - canAttack(Player, (Neutral NPCs, Hostile NPCs)),
        - canUse((Player, Companion NPCs), Wagon Inventory),
        - canTraverse(Player, Environment),
        - canRestAt(Player, (Environment, Forts)),
        - canAmbush(Hostile NPCs, Player)
        - canSteal(Hostile NPCs, Wagon)
        - canDamage((Environment, Hostile NPCs), Wagon)
        - takeFood((Player, Companion NPCs), Wagon Inventory)
        - takeClothes((Player, Companion NPCs), Wagon Inventory)
        - useRepairs(Wagon, Wagon Inventory)
    - *Constraints*:
        - Player and Companion NPCs must consume at least 1 pound of food per day.
        - Player and Companion NPCs must 2 layers of clothing during the winter

        - Player and companion NPCs who fail to meet the constraints listed will die from starvation or from temperatures.
        - The player must travel 2000 miles to journey from Missori to Oregon to complete the game.
        - The player must be alive to finish the game.
    - *Artifacts*: DSL
        - IF dayStarted(player)

          THEN giveChoices(Player)
        - IF lookingToTrade(player)

          THEN tradePrompt(player, Neutral NPCs)
        - IF dayOngoing(player)

          THEN getActions(environment)
        - IF canAttack(Hostile NPCs)

          THEN attackPlayer(Player, Hostile NPCs)
        - IF dayPassed(player) AND dead(player)

          THEN gameOver(player)
        - IF dayPassed(player) AND alive(player) AND foodAvailable(Wagon Inventory)

          THEN takeFood(player, Wagon Inventory)

          ELSE gameOver(player)
        - IF dayPassed(Companion NPCs) AND alive(Companion NPC) AND foodAvailable(Wagon Inventory)

          THEN takeFood(Companion NPCs, Wagon Inventory)

          Else starvationDeath(Companion NPCs)
        - IF isWinter(Environment) AND NOT wearingWinterClothes(player) AND winterClothesAvaliable(Wagon Inventory)

          THEN takeClothes(Player, Wagon Inventory)

```
ELSE gameOver(player)
```

-
  ```
  IF isWinter(Environment) AND NOT
  wearingWinterClothes(Companion NPCs) AND
  winterClothesAvaliable(Wagon Inventory)
  THEN takeClothes(Companion NPCs, Wagon Inventory)
  ELSE winterDeath(Companion NPCs)
  ```

- Section 1.3 – Inference & Reasoning Strategy
  - *Method*: Player/Environment driven gameplay-loops. The player will store the player state in the journey as well as the current gamestate (aka starting day out, ongoing day, end of day). The environment will store the things at play outside of the Player's control (Neutral NPCs, Hostile NPCs, Weather, Places the player encounters). The Environment will take a play in the starting day out phase and ongoing day phase where it will determine who the player's day will progress.
  - *Conflict Resolution*: Overall the code will be structured so that player actions take precedence over any NPC actions. NPC actions will happen in the ongoing day phase where the player can only react to their actions. This way conflict can be safely avoided by managing the game state.
  - *Explanation*:
    - Player died due to lack of food: Starvation
    - Player died due to lack of clothes: Hypothermia
    - Companion NPC died due to lack of food: Starvation
    - Companion NPC died due to lack of clothes: Hypothermia
    - Player failed to make the trade: Player did not have the eligible items required
  - *Performance*:
    - By resorting the game operations on a state per state basis, operations can safely be ignored based on the current game state. Operations can also be performed incrementally due to the day to day nature of the game where each day the player is prompted.

- Section 1.4 — Knowledge Acquisition & Validation Plan
  - *<u>Sources</u>*: Historical information about the Oregon Trail, Existing Oregon Trail-style games for ideas on mechanics and events, Class materials on CLIPS and rule-based systems
  - *<u>Elicitation</u>*: Break down the game into simple rules, translate the game mechanics into CLIPS facts and rules, then iterate on those rules by testing small pieces at a time
  - *<u>Validation</u>*: Create test scenarios and check that the correct rules fire; Test full gameplay runs to make sure the game loop works and verify win/loss conditions
  - *<u>Governance</u>*: Keep versions of the knowledge base as changes are made; update rules based on testing and feedback; final review before submission to ensure rules are consistent and correct
- Section 1.5 — System Architecture & Integration
  - *<u>Components</u>*:
    - Inference Engine: CLIPS engine that runs rules and updates the game state
    - Knowledge Base: Stores facts and rules in CLIPS
    - Game Loop Controller: Controls the flow of the game
    - User Interface: Text-based input and output system for player decisions
  - *<u>Integrations</u>*: No external systems required; Runs locally using CLIPS
  - *<u>Nonfunctional</u>*: Response time should be fast since rules fire almost instantly; System should not crash during gameplay; Rules should be easy to read and modify; Output should be clear and understandable to the player
  - *<u>Deployment</u>*: Runs as a local CLIPS program; Can be executed through command line or a simple interface; no network or database required