Ryan Poppe CS6300 Project Checkpoint 1
November 10, 2025

Garden Design Assistant - ReAct Agent Design Brief

1. PEAS Analysis

Performance Measures

- **Accuracy of plant recommendations**: Compatibility with zipcode climate (USDA hardiness zone)
- **Planting schedule feasibility**: Alignment with local frost dates and growing seasons
- **Space optimization**: Efficient use of available planter space without overcrowding
- **Goal achievement**: Success rate in meeting user's stated objectives (food production, aesthetics, pollinators, etc.)
- **Plant health predictions**: Companion planting benefits, avoiding known antagonistic pairings
- **User satisfaction**: Ease of following recommendations, clarity of visual outputs
- **Constraint adherence**: Respect for user's whitelist/blacklist preferences

Environment

- **User inputs**: Zipcode, planter dimensions (quantity, size, shape), shade levels (full sun, partial, full shade), growing goals, plant preferences
- **External data sources**: USDA plant hardiness zones, frost date databases, plant compatibility matrices, growing requirement databases
- **Time-sensitive factors**: Current date (affects planting schedule), seasonal windows
- **Static constraints**: Physical space limitations, immutable environmental factors (climate zone)

Actuators (Tools/Actions)

The agent interacts with the environment through these key tools:

1. **Climate Data Retrieval**: Fetch hardiness zone and frost dates
2. **Plant Database Query**: Search plants by requirements (sun, water, space, zone)
3. **Compatibility Checker**: Validate companion planting relationships
4. **Space Calculator**: Compute layout given planter dimensions and plant spacing needs
5. **Schedule Generator**: Create planting timeline based on frost dates and plant maturity
6. **Visualization Engine**: Generate SVG/PDF garden layout diagrams

7. **Interactive UI State Manager**: Handle user feedback loops and plan modifications

Sensors (Inputs)

- Structured form data from user (zipcode, measurements, preferences)
- Database query results (plant characteristics, climate data)
- User feedback during interactive refinement
- Validation results from compatibility and space calculations

---

2. Environment Properties

Observable vs Partially Observable

Partially Observable: The agent cannot directly observe:

- Actual microclimate conditions (urban heat islands, wind patterns, local soil quality)
- User's gardening experience level
- Future weather patterns that may affect success
- Actual available sunlight hours (user provides approximate shade level)

The agent observes through:

- User-provided inputs (potentially inaccurate)
- Database queries (historical/average data)

Deterministic vs Stochastic

Stochastic:

- Plant growth outcomes are probabilistic (weather, pests, disease, user care)
- Database queries may return varying results based on data updates
- User preferences may be ambiguous or change during interaction
- Climate data reflects historical averages, not guaranteed future conditions

Episodic vs Sequential

Sequential:

- Garden planning involves temporal dependencies (planting succession, crop rotation considerations)

- Early decisions (spring plantings) affect later options (summer/fall plantings)
- User interactions build on previous responses
- The planting schedule is inherently sequential across seasons

Static vs Dynamic

Dynamic:

- User may modify requirements during conversation
- Seasonal appropriateness changes with current date
- Database information may be updated between queries
- Interactive refinement means the goal state evolves

Discrete vs Continuous

Hybrid:

- **Discrete**: Plant selections (categorical choices), planter quantities, planting dates
- **Continuous**: Planter dimensions, spacing measurements, shade percentages, growing season durations

Single-agent vs Multi-agent

Single-agent: One AI agent assists one user, though it coordinates multiple tools

Known vs Unknown

Partially Unknown:

- Plant database may be incomplete
- Companion planting science has some uncertainty
- User's success depends on unknown factors (gardening skill, maintenance consistency)
- Climate change introduces uncertainty in historical zone data

---

3. Tool Specifications

Tool 1: get_climate_data

Description: Retrieves USDA hardiness zone and average last/first frost dates for a given

zipcode.

Inputs:

- `zipcode` (string): 5-digit US postal code

Outputs:

- `hardiness_zone` (string): e.g., "7b" (USDA zone)
- `last_spring_frost` (date): Average last frost date in spring
- `first_fall_frost` (date): Average first frost date in fall
- `growing_season_days` (integer): Days between frost dates

Error Handling:

- Invalid zipcode Request validation and re-prompt
- Zipcode not found Request nearest major city or alternative zipcode
- API timeout Retry with exponential backoff, fallback to cached data
- Data unavailable Ask user if they know their zone manually

Tool 2: query_plant_database

Description: Searches plant database for species matching specified growing requirements and constraints.

Inputs:

- `plant_type` (string, optional): "vegetable", "herb", "flower", "fruit"
- `hardiness_zone` (string): Target USDA zone
- `sun_requirement` (enum): "full_sun", "partial_shade", "full_shade"
- `water_needs` (enum): "low", "medium", "high"
- `space_category` (enum): "small", "medium", "large" (based on planter size)
- `growing_goal` (string, optional): "food_production", "pollinator_support", "aesthetics", "herbs"
- `whitelist` (list[string], optional): Specific plants user wants
- `blacklist` (list[string], optional): Plants to exclude

Outputs:

- `plant_list` (list[dict]): Array of plant objects containing: - `common_name` (string) - `scientific_name` (string) - `spacing_inches` (float) - `days_to_maturity` (integer) - `planting_method` (enum): "seed", "transplant", "both" - `height_inches` (float) -

`spread_inches` (float) - `continuous_harvest` (boolean) - `companion_plants` (list[string]) - `antagonist_plants` (list[string])

Error Handling:

- No matches found → Relax constraints incrementally (e.g., expand space_category)
- Conflicting requirements → Inform user of the conflict and ask for prioritization
- Whitelist plant unavailable for zone → Suggest similar alternatives or warn of difficulty
- Database connection error → Fall back to a curated subset of common plants

---

Tool 3: check_companion_compatibility

Description: Validates companion planting relationships between selected plants to optimize growth and pest management.

Inputs:

- `plant_a` (string): Common or scientific name
- `plant_b` (string): Common or scientific name

Outputs:

- `relationship` (enum): "beneficial", "neutral", "antagonistic"
- `reason` (string): Explanation (e.g., "Basil repels aphids that harm tomatoes")
- `confidence` (float): 0.0-1.0 (based on scientific consensus)

Error Handling:

- Plant name not recognized → Attempt fuzzy matching, request clarification
- Insufficient data → Return "neutral" with low confidence, note in explanation
- Contradictory sources → Present most authoritative source with confidence score

---

Tool 4: calculate_planter_layout

Description: Computes spatial arrangement of plants within given planter dimensions, accounting for spacing requirements and companion preferences.

Inputs:

- `planter_dimensions` (dict): - `length_inches` (float) - `width_inches` (float) - `shape` (enum): "rectangular", "circular", "raised_bed"
- `selected_plants` (list[dict]): Each containing plant details from query_plant_database
- `optimization_goal` (enum): "maximize_yield", "maximize_diversity", "aesthetic_arrangement"

Outputs:

- `layout` (list[dict]): Plant placement instructions: - `plant_name` (string) - `quantity` (integer) - `position_x` (float): Inches from origin - `position_y` (float): Inches from origin - `notes` (string): e.g., "Place near edge for easy harvest"
- `utilization_percentage` (float): Space efficiency
- `warnings` (list[string]): Potential issues (overcrowding, incompatible adjacencies)

Error Handling:

- Insufficient space Suggest removing plants or choosing smaller varieties
- Impossible layout Propose increasing planter size or splitting across multiple planters
- Conflicting constraints Prioritize based on optimization_goal

---

Tool 5: generate_planting_schedule

Description: Creates a temporal planting plan based on frost dates, plant maturity times, and succession planting opportunities.

Inputs:

- `plants` (list[dict]): Selected plants with maturity data
- `frost_dates` (dict): Last spring and first fall frost
- `current_date` (date): For schedule relevance
- `succession_planting` (boolean): Whether to plan for multiple harvests

Outputs:

- `schedule` (list[dict]): - `plant_name` (string) - `action` (enum): "start_indoors", "direct_sow", "transplant" - `date_range_start` (date) - `date_range_end` (date) - `expected_harvest` (date, optional) - `notes` (string): Special instructions
- `timeline_svg` (string, optional): SVG Gantt-style chart

Error Handling:

- Current date past optimal planting Suggest alternatives for remaining season

- Conflicting schedules → Prioritize by user's stated goals
- Short growing season → Recommend season extension techniques (row covers, cold frames)

Tool 6: generate_garden_visualization

Description: Creates visual representation of garden layout as SVG or PDF.

Inputs:

- `layout_data` (dict): Output from calculate_planter_layout
- `planter_config` (dict): Dimensions and shape details
- `format` (enum): "svg", "pdf"
- `style` (enum): "top_down", "isometric", "labeled_diagram"

Outputs:

- `visualization` (binary/string): SVG XML string or PDF binary data
- `file_path` (string): Location of generated file
- `legend` (dict): Plant symbols and color coding

Error Handling:

- Rendering failure → Fall back to simpler ASCII-art style text representation
- File system error → Return base64 encoded data directly
- Invalid format request → Default to SVG

Tool 7: interactive_plan_modifier

Description: Handles user feedback and iterative refinement of the garden plan.

Inputs:

- `current_plan` (dict): Complete garden plan (layout + schedule)
- `user_feedback` (string): Natural language modification request
- `feedback_type` (enum): "add_plant", "remove_plant", "adjust_position", "change_timing", "general_question"

Outputs:

- `updated_plan` (dict): Modified plan reflecting changes
- `change_summary` (string): Explanation of what changed and why
- `new_considerations` (list[string]): Ripple effects (e.g., "Moving tomatoes affects basil companion benefit")

Error Handling:

- Ambiguous request Ask clarifying questions
- Infeasible modification Explain constraint violation and suggest alternatives
- Cascading conflicts Present options for resolving (with trade-offs)

---

4. Agent Architecture

Framework

LangGraph with ReAct Pattern

Reasoning:

- **ReAct (Reasoning + Acting)**: Combines chain-of-thought reasoning with tool use, ideal for complex multi-step planning
- **LangGraph**: Provides explicit state management for iterative refinement and branching logic
- Supports cyclic workflows necessary for feedback loops and plan optimization

Model Selection

- **Primary LLM**: GPT-4 or Claude 3.5 Sonnet - Strong reasoning capabilities for complex horticultural logic - Good tool-calling performance - Ability to handle nuanced user preferences
- **Fallback Model**: GPT-3.5-turbo for simpler queries (cost optimization)
- **Specialized Models**: - **Embedding model**: text-embedding-3-small (for plant database semantic search) - **Vision model** (future): For analyzing user photos of existing garden spaces

Reasoning Loop

1. OBSERVE: Parse user input and current state
2. REASON: Analyze requirements, identify information gaps
3. PLAN: Determine which tools to call and in what sequence
4. ACT: Execute tool calls (may be parallel where independent)

5. REFLECT: Evaluate tool outputs, identify conflicts/issues
6. DECIDE: Continue gathering info, refine plan, or present to user
7. [If user provides feedback] Return to step 1

Orchestration Strategy

Hybrid: Model-Driven with Scripted Guardrails

Model-Driven Components:

- Tool selection based on reasoning about current information gaps
- Natural language parsing of user feedback
- Resolving ambiguities in user preferences
- Creative problem-solving when constraints conflict

Scripted Components:

- **Hard constraints**: Always check climate data before plant selection
- **Validation layer**: Automated compatibility checking before finalizing
- **Sequential dependencies**: Schedule generation must follow layout calculation
- **Safety checks**: Prevent toxic plant combinations, respect blacklist

State Management:

```
class GardenPlanState:
    user_requirements: UserInput
    climate_data: ClimateInfo | None
    candidate_plants: List[Plant]
    compatibility_matrix: Dict[Tuple[str, str], Relationship]
    layout: PlantLayout | None
    schedule: PlantingSchedule | None
    conversation_history: List[Message]
    validation_status: ValidationResult
```

Workflow Phases

Phase 1: Discovery (Scripted sequence)

1. Collect all user inputs
2. Validate inputs (zipcode format, measurement units)
3. Call get_climate_data (required)
4. Set up constraints dictionary

Phase 2: Plant Selection (Model-driven with iteration)

1. Reason about goals tool query parameters

2. Call query_plant_database
3. Evaluate results → sufficient variety? meet goals?
4. If insufficient → adjust constraints and retry
5. For each plant pair → check_companion_compatibility
6. Build compatibility matrix

Phase 3: Layout Planning (Hybrid)

1. Group plants by planter (model decides based on requirements)
2. For each planter → calculate_planter_layout (scripted call)
3. Validate against space constraints (scripted)
4. If invalid → model reasons about which plants to adjust
5. Iterate until valid layout achieved

Phase 4: Scheduling (Scripted with model explanation)

1. Call generate_planting_schedule (scripted)
2. Model generates natural language explanation of timeline
3. Highlight critical dates and succession opportunities

Phase 5: Visualization (Scripted)

1. Call generate_garden_visualization with user's preferred format
2. Package final plan with all components

Phase 6: Refinement (Model-driven loop)

1. Present plan to user
2. If feedback → interactive_plan_modifier analyzes request
3. Model determines which components need regeneration
4. Return to appropriate phase based on modification scope

Error Recovery Strategy

- **Validation failures**: Explicitly reason about violated constraints, propose relaxations
- **Tool failures**: Attempt retry, fall back to alternative data sources, or ask user for manual input
- **Conflicting goals**: Present trade-offs transparently, ask user to prioritize
- **Dead ends**: Backtrack to last decision point, explore alternative branch

---

5. Evaluation Plan

Test Cases

Test Case 1: Basic Vegetable Garden (Happy Path)

Input:

- Zipcode: 94102 (San Francisco, Zone 10a)
- Planters: 1 raised bed, 4ft x 8ft
- Shade: Full sun (6+ hours)
- Goal: Food production
- Preferences: Likes tomatoes, no eggplant

Expected Behavior:

1. Correctly identifies Zone 10a, frost-free climate
2. Suggests companion trio: tomatoes, basil, marigolds
3. Includes space for lettuce or greens (quick harvest)
4. Layout respects 24" tomato spacing
5. Schedule shows year-round growing options
6. Visualization clearly shows plant positions

Success Metrics:

- All plants compatible with Zone 10a
- Layout utilization 75-90%
- No antagonistic pairings
- Schedule within 2 weeks of optimal dates
- Eggplant not included

Test Case 2: Shade Garden with Constraints

Input:

- Zipcode: 10001 (NYC, Zone 7b)
- Planters: 3 containers, 18" diameter each
- Shade: Full shade (<2 hours sun)
- Goal: Herbs for cooking
- Preferences: None

Expected Behavior:

1. Identifies Zone 7b with frost dates
2. Selects shade-tolerant herbs (parsley, cilantro, mint, chives)
3. Warns that basil (sun-loving) won't thrive

4. Places one herb type per container (mint is invasive)
5. Schedule accounts for cool-season preference of cilantro

Success Metrics:

- No full-sun plants suggested
- One plant per container (appropriate for size)
- Mint isolated to prevent spread
- At least 3 different herbs offered
- Planting dates match Zone 7b spring/fall windows

Test Case 3: Conflicting Requirements

Input:

- Zipcode: 55401 (Minneapolis, Zone 5a)
- Planters: 2 small pots, 10" x 10" each
- Shade: Partial (3-4 hours)
- Goal: Grow watermelons and pumpkins
- Preferences: None

Expected Behavior:

1. Agent reasons that large vining plants won't fit in 10" pots
2. Explains constraint violation clearly
3. Offers alternatives: - Suggest larger planters (24"+ diameter) - Suggest compact varieties (bush cucumber instead) - Suggest different goals (herbs, greens)
4. If user insists, provide warning about poor success likelihood

Success Metrics:

- Agent explicitly identifies size mismatch
- Provides at least 2 alternative paths
- Maintains helpful tone despite infeasibility
- Does not generate invalid layout

Test Case 4: Succession Planting

Input:

- Zipcode: 78701 (Austin, Zone 9a)
- Planters: 1 raised bed, 3ft x 6ft
- Shade: Full sun
- Goal: Maximize yield over seasons
- Preferences: Loves salads, peppers

Expected Behavior:

1. Plans multi-season rotation
2. Spring: lettuce, spinach (cool season)
3. Summer: peppers, heat-tolerant greens (amaranth)
4. Fall: replant lettuce after peppers harvested
5. Companion plants peppers with basil

Success Metrics:

- Schedule includes at least 2 plantings per space
- Cool vs warm season crops appropriate
- Total growing days maximized (>200 days coverage)
- Harvest gaps < 3 weeks

Test Case 5: Interactive Refinement

Input (Initial):

- Basic plan generated for Zone 8a, 4x4 bed, full sun, vegetables

Feedback Loop:

- User: "I don't like kale, swap it for something else"
- Agent: Removes kale, suggests collards or swiss chard (similar growth habit)
- User: "Add more flowers for pollinators"
- Agent: Adds marigolds and zinnias to edges
- User: "When do I plant the tomatoes?"
- Agent: References schedule, explains start indoors 6 weeks before last frost

Expected Behavior:

1. Handles each modification request appropriately
2. Maintains plan coherence (no broken companion relationships)
3. Provides context-appropriate responses (tool calls vs direct answers)
4. Updates visualization after each change

Success Metrics:

- Kale removed, appropriate substitute added
- Flowers integrated without displacing food crops
- Direct answer to date question (no unnecessary tool calls)
- Final plan internally consistent

Success Metrics

Functional Correctness (Must-Pass)

- **Zone Accuracy**: 100% of plants compatible with user's hardiness zone
- **Space Validity**: 100% of layouts fit within planter dimensions
- **Safety**: 0% of antagonistic plant pairings in final plan
- **Constraint Adherence**: 100% respect for blacklist items

Quality Indicators (Target Ranges)

- **Space Utilization**: 70-90% (efficient but not overcrowded)
- **Companion Benefits**: >60% of plant pairs have beneficial or neutral relationships
- **Seasonal Coverage**: Growing season utilization >70%
- **Goal Alignment**: >80% of plants directly support stated goal

User Experience Metrics

- **Interaction Efficiency**: <5 turns to reach satisfactory plan (90th percentile)
- **Clarity**: User understands reasoning in >90% of agent explanations (user study)
- **Actionability**: Users can follow schedule without external research (user feedback)
- **Visual Quality**: Visualization is interpretable without extensive text (user rating >4/5)

Robustness Measures

- **Error Handling**: Graceful degradation in 100% of tool failures
- **Ambiguity Resolution**: Agent seeks clarification rather than guessing (detect in >80% of ambiguous inputs)
- **Constraint Relaxation**: When overspecified, agent proposes logical compromises (manual review)

Testing Methodology

Automated Tests:

- Unit tests for each tool with edge cases
- Integration tests for scripted workflow sequences
- Regression suite for known bug scenarios

Human Evaluation:

- Expert review (master gardeners validate botanical accuracy)
- Novice user study (5 participants, diverse climate zones)
- A/B test against static rule-based planner

Continuous Monitoring:

- Log tool call patterns (detect inefficient reasoning)
- Track user satisfaction ratings per session
- Monitor constraint violation frequency

---

6. Implementation Considerations

Data Requirements

- **Plant Database**: Minimum 200 common plants with comprehensive attributes
- **Climate Database**: USDA zone mappings for all US zipcodes
- **Companion Matrix**: Scientifically-backed relationships (academic sources preferred)

UI/UX Design

- **Progressive Disclosure**: Start with simple form, reveal advanced options
- **Real-time Validation**: Warn about issues as user types (e.g., invalid zipcode)
- **Visualization Interactivity**: Allow drag-drop to manually adjust layout (feeds back to agent)
- **Mobile-Friendly**: Schedule and care reminders via mobile app

Scalability & Performance

- **Caching**: Store climate data and common plant queries
- **Async Tool Calls**: Parallel execution where dependencies allow
- **Streaming Responses**: Show reasoning steps as they happen (builds trust)

Future Enhancements

- **Photo Analysis**: "Here's my space" → computer vision extracts dimensions, shade
- **IoT Integration**: Soil sensor data refines watering recommendations
- **Community Features**: Share plans, fork others' successful gardens
- **Multi-Season Planning**: Year-over-year crop rotation strategies
- **Pest/Disease Diagnosis**: "Why are my tomato leaves curling?" → visual + contextual diagnosis

---

Summary

This ReAct agent architecture balances automation with user agency, leveraging LLM reasoning for complex horticultural decisions while maintaining scripted guardrails for critical validations. The tool ecosystem provides comprehensive coverage from climate research through visual output, with error handling strategies for real-world messiness. The evaluation plan ensures both technical correctness and user satisfaction, positioning the agent as a knowledgeable gardening partner rather than a black-box recommendation engine.