8th Grade: Ideation & Skill Foundations

Goal: Explore ideas, learn basics, and build habits.

Writing / Creative:

- Brainstorm themes: trust, self-growth, and romantasy elements.
- Write short character sketches, world-building notes, or mini-scenes.
- Analyze existing fantasy stories and note how authors explore trust or personal growth.

Coding / STEM:

- Learn fundamentals: Python basics, JavaScript, or beginner web development.
- Create mini-experiments: text-based “choose your path” games or small interactive scripts.
- Learn basic logic structures, variables, arrays, and simple user input handling.

Supporting Activities:

- Journaling ideas and reflections.
- Join coding or writing clubs.
- Track progress for future essays/portfolio.

9th Grade: First Prototype

Goal: Build a minimal viable product (MVP) that demonstrates both coding and story.

Creative:

- Draft first chapters of your story with branching paths (2–3 key decision points).
- Incorporate trust-building scenarios in the plot.

STEM / Coding:

- Build MVP app with:
    - Branching story logic using if/else statements.
    - Basic tracking of "trust level" or decision outcomes.
- Test with friends/family to gather feedback.
- Start documenting code structure on GitHub.

Other Activities:

- Small competitions or school showcases for coding/writing.
- Reflect on what's working and what can be improved.

10th Grade: Expansion & Early Impact

Goal: Make the app more interactive and your story deeper.

Creative:

- Expand story arcs and character development.
- Add reflective prompts for users ("Why would you forgive this character?").
- Include multiple story branches and outcomes.

STEM / Coding:

- Upgrade to web-based app: HTML/CSS + JS or Python Flask.
- Implement:
    - Visual feedback (charts, simple graphics).
    - Tracking of user decisions for each session.
- Begin small-scale external beta testing (classmates, friends).
- Document challenges solved (logic, bug fixes, design decisions).

Other Activities:

- Enter coding/interactive storytelling competitions.
- Publish excerpts of the story online for feedback.

11th Grade: Polishing & Measurable Impact

Goal: Make the app polished, functional, and impactful; prepare for competitions and portfolio.

Creative:

- Finalize story content for multiple endings.
- Enhance narrative depth and emotional reflection.
- Add small interactive features: mini-games, optional side stories, or character journals.

STEM / Coding:

- Add advanced features:
    - Dynamic branching paths that adapt based on prior choices.

    - Optional sentiment analysis of user input (Python libraries like NLTK or TextBlob).
    - More sophisticated visualizations of user decisions or “trust metrics.”
- Public beta release; track engagement metrics.
- Prepare detailed documentation for GitHub/portfolio showing problem-solving and coding skills.

Other Activities:

- Leadership: Run small workshops or club activities around interactive storytelling or coding.
- Link project to IB Extended Essay or school research project.

12th Grade: Final Launch & College Prep

Goal: Showcase mastery and impact for college applications.

Creative:

- Publish the final story digitally (interactive web app or Twine) or small print edition.
- Highlight reflective prompts and trust-building lessons in the app.

STEM / Coding:

- Fully functional interactive story app with polished UI/UX.
- Advanced logic, analytics, and optional AI-enhanced reflections.
- Portfolio-ready: GitHub repository, screenshots, and user engagement metrics.

College Application:

- Essays:
    - Emphasize interdisciplinary curiosity (coding + writing + psychology/philosophy).
    - Explain problem-solving, creativity, and self-growth through project.
- Recommendations: Teachers can comment on initiative, technical skill, and creative depth.

Key Principles for STEM + Arts Balance

1. **Document rigor:** Every coding challenge solved counts as STEM evidence.
2. **Tie logic to narrative:** Show how coding decisions enhance the story (branching paths, trust calculations).
3. **Measure impact:** Collect user feedback and visualize results (graphs, stats).
4. **Iterative growth:** Start simple, layer complexity each year — ensures quality without burnout.
5. **Portfolio-ready:** Keep all drafts, code, and reflections organized for essays and showcase.

1. Branching Story Logic (Core Coding Challenge)

What it is:

- The story changes depending on the choices the user makes.
- Each choice affects the narrative and potentially a “trust score” for characters.
- Example:
    - User chooses to forgive a character → trust score +10, story path A

    - User chooses to distance themselves → trust score -5, story path B

Why it shows intelligence:

- Requires **algorithmic thinking**: you must track variables, states, and dependencies.
- Involves **conditional logic, nested decisions, and flow control** — core programming skills.
- Planning the structure of all possible paths is like designing a **mini software system**.

How to make it stronger:

- Use **data structures** like arrays, dictionaries, or trees to manage choices and outcomes.
- Implement **dynamic branching** where earlier choices affect later story events in non-linear ways.
- Optional: simulate "trust probability" — e.g., past behavior affects likelihood of future outcomes.

---

2. Trust/Emotion Tracking & Analytics

What it is:

- Each choice the user makes is tracked in a system (like a trust meter, score, or statistics).
- Can visualize results: charts showing choices, trust levels, or trends over multiple sessions.

Why it shows intelligence:

- Shows ability to **collect, process, and visualize data**.

- Demonstrates **analytical thinking** — you're not just coding a story, you're turning user interactions into meaningful metrics.

- Optional: allows for **pattern recognition or predictive features**, like suggesting likely outcomes or showing "how different choices affect trust."

How to make it stronger:

- Add small data analysis: e.g., use Python libraries like Matplotlib, Pandas, or Plotly to visualize user behavior.

- Optional: implement **basic AI/sentiment analysis** — analyze free-text responses and adjust story dynamically.

---

3. Interactive Features / Mini-Simulations

What it is:

- Extra layers beyond basic choices: mini-games, quizzes, or decision simulations embedded in the story.

- Example: a short "trust-building challenge" where users solve a puzzle to gain points toward the trust score.

Why it shows intelligence:

- Demonstrates **problem-solving and coding versatility**.

- Adds **computational thinking**: you design logic, rules, and feedback systems.

- Shows you can integrate **systems thinking** (combining narrative, user input, and algorithmic outcomes).

---

4. Data Structures & Technical Design

What it is:

- Internally, your app will need **variables, arrays/lists, dictionaries, objects**, and possibly a database to track user sessions.
- Story branching can be implemented as a **decision tree or graph**, with nodes representing story events and edges representing choices.

Why it shows intelligence:

- Shows knowledge of **software architecture** — how you organize code efficiently.
- Solving complex branching paths demonstrates **logical and structural thinking**.
- This is **undeniable STEM evidence**, especially if documented in GitHub or portfolio.

---

5. Integration of Psychology/Philosophy

What it is:

- The narrative itself explores trust, self-awareness, and ethical decision-making.
- You can include prompts like: “Why did you choose forgiveness?” or “How does this decision reflect your values?”

Why it shows intelligence:

- Demonstrates **higher-order thinking** — understanding human behavior, moral reasoning, and emotional consequences.
- Shows **interdisciplinary curiosity**, which Stanford highly values.
- When combined with coding, it’s a **rare mix of STEM + humanities**, proving intellectual versatility.

---

6. How to Make the Project “Stanford-Ready”

1. **Document everything:**
    - GitHub repo with code, version control, commit history, and readme explaining logic.
    - Flowcharts for branching story logic.
    - Screenshots or videos showing user interaction and analytics.
2. **Measure impact:**
    - Track beta users and their choices.
    - Show engagement metrics (e.g., how many tried multiple endings, average trust scores).
3. **Layer complexity gradually:**
    - 8th–9th: text-based, small branching.
    - 10th: web app with simple analytics and visuals.
    - 11th: advanced branching, adaptive paths, trust metrics.
    - 12th: polished UI/UX, optional AI/sentiment analysis, mini-games.
4. **Tie to essays and recommendations:**
    - Show how your coding and creative work combine **problem-solving, creativity, and reflection**.
    - Highlight learning from bugs, user feedback, and story design challenges.

Bottom line

This project can **clearly show intelligence and STEM skill**, while also showcasing your creativity and emotional insight. It's not just "arts-heavy" — it's **interdisciplinary, technically sophisticated, and reflective**.

If you implement:

- Branching logic with data structures
- User analytics / trust metrics
- Optional AI or simulations

1. Overambition / Risk of Burnout

Problem: Project is large: writing, coding, graphics, interactivity, analytics.

Solution:

- **Layer complexity gradually:**
    - 8th Grade → small prototype (text-based, 2–3 decision points).
    - 9th → add branching paths and trust tracking.
    - 10th → graphics, basic visualization.
    - 11th → advanced branching, sentiment analysis, mini-games.
    - 12th → polished UI/UX, final analytics/dashboard.
- **Set small milestones:** Break each year into achievable monthly goals.
- **Prioritize:** Always focus on making one part of the app polished before adding new features.

---

2. Coding & Logic Complexity

Problem: Branching story logic, dynamic metrics, interactive graphics can create bugs.

Solution:

- **Start simple:** Prototype small sections before scaling.
- **Modular code design:** Keep story, graphics, analytics, and user input separate modules.
- **Use version control:** Git/GitHub to track changes, revert bugs, and document progress.
- **Seek guidance:** Ask teachers, mentors, or online coding communities if stuck.
- **Document everything:** Include flowcharts, pseudocode, and comments in your code.

---

3. User Engagement & Data Validity

Problem: Users may give minimal responses or skip prompts.

Solution:

- **Adaptive prompts:** If a user responds vaguely, ask simpler follow-ups or give multiple-choice options.
- **Aggregate metrics:** Don't rely on individual responses; track trends over all users.
- **Optional gamification:** Add small rewards for completing prompts to encourage participation.
- **Use low-effort responses as data:** For example, track the percentage of users who skip or write short answers — that itself is meaningful.

---

4. Time Management

Problem: Completing this complex project could interfere with school, IB/honors, and extracurriculars.

Solution:

- **Integrate milestones into school years:** Make each grade responsible for one major development step.
- **Weekly or monthly schedule:** Dedicate 1–2 focused hours per week to coding/writing for consistency.
- **Avoid multitasking multiple major projects:** Let this app be your "signature project," and keep other activities smaller.

---

5. Balancing STEM + Arts

Problem: Project could lean too heavily on one side (story or coding).

Solution:

- **Every technical feature serves the story:** Trust metrics, analytics, or visualizations must relate to user choices and emotional reflection.
- **Every story element integrates STEM:** Branching logic, interactive graphics, or dashboards should support narrative and reflection.
- **Document both:** In portfolio and essays, clearly show both **creativity and technical problem-solving**.

---

6. Polish / Presentation

Problem: Without polished graphics, UI, and documentation, project may seem amateur.

Solution:

- **Add visuals gradually:** Start with static illustrations → interactive graphics → animations.
- **User testing:** Have friends/classmates test the app for UX feedback.

- **Document all stages:** Screenshots, flowcharts, and version history show thoughtfulness and polish.
- **UI/UX tutorials:** Learn basic design principles for layout, colors, and readability.

---

7. Scalability / Technical Limitations

Problem: Large branching trees and graphics could be difficult to manage or slow performance.

Solution:

- **Limit branching early:** Keep story paths manageable (2–3 per chapter). Expand later.
- **Optimize graphics:** Use lightweight images or vector graphics; compress assets.
- **Modular design:** Allows adding new paths or graphics without breaking old code.
- **Test performance regularly:** Identify slow parts and simplify if needed.

✅ Key Takeaways

- **Start small, grow gradually:** Focus on depth over breadth.
- **Document everything:** Code, design, testing, user feedback.
- **Integrate STEM & arts:** Every feature should serve both logic and creativity.
- **Manage time:** Spread complexity over multiple years.
- **Polish & test:** Make it visually and functionally impressive.