AI Features Integration Plan - IntelectGame V2

This document outlines potential AI features that can be integrated into the IntelectGame V2 platform to enhance functionality and user experience.

Overview

The AI features will be integrated through a new `ai-service` microservice that communicates with various AI APIs. This service will be accessible through the existing API Gateway and will leverage Redis for caching and MinIO for storing generated content.

Proposed AI Services Integration

1. OpenAI API Integration

1.1 Automatic Question Generation

- **Feature**: Generate quiz questions automatically based on a specific theme or topic
- **Use Case**: Administrators can input a topic (e.g., "World History", "JavaScript Basics") and receive a set of questions with multiple-choice answers
- **Implementation**:
  - Endpoint: `POST /ai/generate-questions`
  - Input: `{ topic: string, difficulty: string, count: number, language: string }`
  - Output: Array of questions with correct answers and explanations
- **Benefits**: Reduces manual work for administrators, enables rapid quiz creation

1.2 Answer Analysis & Grading

- **Feature**: Evaluate open-ended answers and provide scores
- **Use Case**: For questions that require text responses, AI evaluates the answer quality
- **Implementation**:
  - Endpoint: `POST /ai/analyze-answer`
  - Input: `{ question: string, answer: string, context: object }`
  - Output: `{ score: number, feedback: string, correctness: number }`
- **Benefits**: Enables more complex question types beyond multiple choice

1.3 Question Suggestions & Variations

- **Feature**: Generate alternative questions or variations of existing questions
- **Use Case**: Help administrators create question banks with similar difficulty levels
- **Implementation**:
  - Endpoint: `POST /ai/suggest-questions`
  - Input: `{ question: string, count: number }`
  - Output: Array of similar questions with variations
- **Benefits**: Helps maintain question diversity while keeping difficulty consistent

1.4 Multi-language Translation

- **Feature**: Automatically translate quizzes and questions to multiple languages
- **Use Case**: Make quizzes accessible to international audiences
- **Implementation**:
  - Endpoint: `POST /ai/translate-quiz`
  - Input: `{ quizId: string, targetLanguages: string[] }`
  - Output: Translated quiz content for each language
- **Benefits**: Expands user base, improves accessibility

1.5 Content Moderation

- **Feature**: Automatically check if questions and answers comply with content policies
- **Use Case**: Ensure user-generated content meets quality and safety standards
- **Implementation**:

    - Endpoint: `POST /ai/moderate-content`
    - Input: `{ content: string, type: string }`
    - Output: `{ approved: boolean, issues: string[], suggestions: string[] }`
- **Benefits**: Maintains content quality, reduces manual moderation

2. Replicate.com / Image Generation APIs

2.1 Question Image Generation

- **Feature**: Generate images for quiz questions based on descriptions
- **Use Case**: Create visual questions without manual image creation
- **Implementation**:
    - Endpoint: `POST /ai/generate-question-image`
    - Input: `{ description: string, style: string, questionId: string }`
    - Output: Image URL stored in MinIO
- **Benefits**: Enhances visual appeal, supports visual learning

2.2 Concept Illustration

- **Feature**: Generate diagrams, charts, or concept illustrations for educational content
- **Use Case**: Create visual aids for complex topics
- **Implementation**:
    - Endpoint: `POST /ai/generate-diagram`
    - Input: `{ concept: string, type: string, format: string }`
    - Output: Illustration URL stored in MinIO
- **Benefits**: Improves understanding of complex topics

2.3 Player Avatar Generation

- **Feature**: Generate personalized avatars for players
- **Use Case**: Create unique visual identities for users
- **Implementation**:
    - Endpoint: `POST /ai/generate-avatar`
    - Input: `{ playerId: string, preferences: object }`
    - Output: Avatar URL stored in MinIO
- **Benefits**: Enhances user engagement and personalization

3. Advanced AI Features

3.1 Personalized Quiz Recommendations

- **Feature**: Suggest quizzes based on player performance and preferences
- **Use Case**: Help players discover relevant content
- **Implementation**:
  - Endpoint: `GET /ai/recommend-quizzes`
  - Input: `{ playerId: string, history: object }`
  - Output: Array of recommended quiz IDs with reasoning
- **Benefits**: Improves user engagement, personalized learning paths

3.2 Performance Analysis

- **Feature**: Analyze player performance and identify weak areas
- **Use Case**: Provide insights to players about their knowledge gaps
- **Implementation**:
  - Endpoint: `GET /ai/analyze-performance`
  - Input: `{ playerId: string, timeRange: string }`
  - Output: `{ strengths: string[], weaknesses: string[], recommendations: string[] }`
- **Benefits**: Helps players focus on areas needing improvement

3.3 Adaptive Difficulty

- **Feature**: Automatically adjust question difficulty based on player performance
- **Use Case**: Provide optimal challenge level for each player
- **Implementation**:
  - Endpoint: `POST /ai/adjust-difficulty`
  - Input: `{ playerId: string, currentPerformance: object }`
  - Output: Recommended difficulty level and question selection
- **Benefits**: Optimizes learning experience, prevents frustration

3.4 Question Quality Scoring

- **Feature**: Evaluate and score the quality of user-generated questions
- **Use Case**: Help administrators identify high-quality questions
- **Implementation**:

    - Endpoint: `POST /ai/score-question-quality`
    - Input: `{ question: object, context: object }`
    - Output: `{ qualityScore: number, feedback: string, suggestions: string[] }`
- **Benefits**: Maintains quiz quality standards

Architecture Integration

New Service: `ai-service`

```
ai-service (Port 3005)
├── OpenAI API integration
├── Replicate.com integration
├── Image generation handling
├── Caching layer (Redis)
└── Storage integration (MinIO)
```

API Gateway Routes

```
/vika-game/api/ai/generate-questions
/vika-game/api/ai/analyze-answer
/vika-game/api/ai/translate-quiz
/vika-game/api/ai/generate-question-image
/vika-game/api/ai/recommend-quizzes
/vika-game/api/ai/analyze-performance
```

Data Flow

1. **Question Generation Flow**:

   ```
   Admin → API Gateway → ai-service → OpenAI API → ai-service → Quiz
   Service → MongoDB
   ```

2. **Image Generation Flow**:

```
Admin → API Gateway → ai-service → Replicate.com → MinIO → ai-service →
Quiz Service
```

3. **Performance Analysis Flow**:

```
Player → API Gateway → ai-service → Game Service (get history) → OpenAI
API → ai-service → Player
```

Implementation Priority

Phase 1 (High Priority)

1. ✅ Question generation based on theme
2. ✅ Content moderation
3. ✅ Question image generation

Phase 2 (Medium Priority)

4. ✅ Answer analysis for open-ended questions
5. ✅ Multi-language translation
6. ✅ Personalized recommendations

Phase 3 (Future Enhancements)

7. ✅ Performance analysis
8. ✅ Adaptive difficulty
9. ✅ Avatar generation

Technical Requirements

Dependencies

- `openai` package for OpenAI API
- `replicate` package for Replicate.com
- `axios` for HTTP requests
- `redis` client for caching
- `minio` client for image storage

Environment Variables

```
OPENAI_API_KEY=your_key_here
REPLICATE_API_TOKEN=your_token_here
AI_SERVICE_PORT=3005
REDIS_HOST=redis
REDIS_PORT=6379
MINIO_ENDPOINT=minio:9000
MINIO_ACCESS_KEY=minioadmin
MINIO_SECRET_KEY=minioadmin
```

Caching Strategy

- Cache generated questions for 24 hours
- Cache image generation results permanently
- Cache recommendations for 1 hour per user
- Use Redis keys: `ai:questions:{topic}`, `ai:images:{hash}`, `ai:recommendations:{playerId}`

Cost Considerations

OpenAI API

- Question generation: ~$0.01-0.05 per quiz (10-20 questions)
- Answer analysis: ~$0.001 per answer
- Translation: ~$0.01 per quiz per language

Replicate.com

- Image generation: ~$0.01-0.03 per image

- Model-dependent pricing

Optimization

- Implement aggressive caching to reduce API calls
- Batch requests when possible
- Use rate limiting to control costs
- Monitor usage with Prometheus metrics

Security Considerations

1. **API Key Management**: Store keys securely in environment variables
2. **Rate Limiting**: Implement rate limits per user/admin
3. **Content Filtering**: Always moderate AI-generated content
4. **Data Privacy**: Ensure AI services comply with data protection regulations
5. **Cost Monitoring**: Set up alerts for unexpected API usage spikes

Monitoring & Metrics

Metrics to Track

- AI API call count and latency
- Generation success/failure rates
- Cost per operation
- Cache hit rates
- User engagement with AI features

Grafana Dashboard

- Create dashboard: `/d/ai-service-monitoring/ai-monitoring`
- Track API usage, costs, and performance

Example Use Cases

Use Case 1: Quick Quiz Creation

```
Admin → Select topic "JavaScript" → Generate 20 questions → Review & edit → Publish
Time saved: 2-3 hours → 5 minutes
```

Use Case 2: Visual Question Enhancement

```
Admin → Add question → Describe image needed → AI generates image → Auto-attach to question
Enhancement: Text-only → Rich visual content
```

Use Case 3: Personalized Learning

```
Player → Complete quiz → AI analyzes performance → Recommends next quiz → Suggests study topics
Benefit: Guided learning path
```

Future Enhancements

1. **Voice Questions**: Generate audio questions using text-to-speech
2. **Video Generation**: Create short educational videos for quiz topics
3. **Chatbot Assistant**: AI tutor to help players understand concepts
4. **Automated Quiz Creation**: Full quiz generation from a single topic input
5. **Competitive Analysis**: Compare player performance against AI benchmarks

---

Last Updated: January 2026 **Status**: Planning Phase - Ready for Implementation