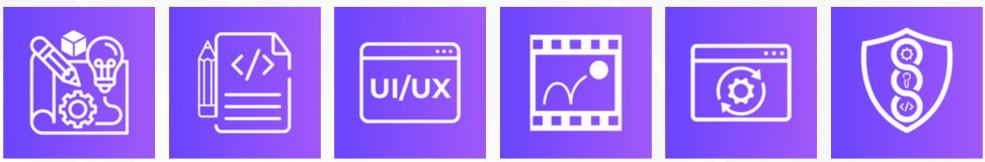
AKYLADE®

Certified C# Developer™ (A/CCSD™) Exam Objectives

Exam Code: CSD-001



AKYLADE

EXAM DETAILS

EXAM NAME	AKYLADE® Certified C# Developer™ (A/CCSD™)
EXAM CODE	CSD-001
QUESTION ITEMS	50 questions
DURATION	90 minutes
PASSING SCORE	700 points on a scale of 100 to 900 points

EXAM DOMAINS

	DOMAIN	EXAM COVERAGE
1.0	Design and Prototyping	16%
2.0	Programming and Scripting	26%
3.0	User Interface (UI) and User Experience (UX)	14%
4.0	Physics, Animation, and Effects	16%
5.0	Optimization and Performance	14%
6.0	DevSecOps	14%

IMPORTANT NOTE:

AKYLADE® continuously reviews and updates exam content to ensure that exams remain current, relevant, and secure. When necessary, AKYLADE® may publish updated exams based on the existing exam objectives, but all related exam preparation materials will remain valid during a given exam's lifecycle. Under each objective, a list of bulleted examples has been provided to aid candidates during their studies. These lists are not exhaustive and serve as examples of technologies, processes, regulations, or tasks related to the relevant objective. Other examples may appear on the exam, even if they are not listed directly under the objective in this document.

DOMAIN 1: Design and Prototyping (16%)

Candidates must be able to plan, document, and prototype applications using wireframes, game design documents, and iterative user feedback processes.

1.1 Given a scenario, plan and document an application while incorporating user feedback to refine its mechanics.

- Develop the Game Design Documents (GDD)
  - Define the Scope of Work (SOW)
- Develop flowcharts
  - Gameplay flow
  - Game logic
  - User experience
- Develop wireframes
  - User interface
  - Game experience
- Playtest to gain feedback and improve the design
- Iterate level designs

1.2 Given a scenario, utilize prototypes to efficiently organize scenes, place components, and build basic 2D/3D maps and character prototypes in Unity.

- Rapidly prototype core mechanics
- Create and iterate interactive prototypes
- Set up and organize scenes efficiently
- Utilize Unity's Scene View for placement
  - Objects
  - Components
- Create and build prototypes
  - Characters
  - Objects
  - 2D maps
  - 3D maps

1.3 Given a scenario, design and configure multi-platform control schemes and camera perspectives catering to diverse user inputs and device capabilities.

- Create user controls
  - Navigation
  - Camera perspectives
  - Touch/gestures for mobile devices
- Configure camera systems
  - Orthographic
  - Isometric
  - Perspective
  - Cinematic
  - Top-down
  - Side-scrolling
  - Third-person
  - First-person
  - Split-screen
  - VR/AR
- Implement platform-specific input variations
  - PC
  - Mobile
  - VR/AR
  - Console
- Player control schemes

1.4 Given a scenario, develop engaging environments by adjusting lighting, implementing environmental changes, and leveraging Unity's navigation tools for immersive and dynamic level designs.

- Unity's NavMesh
  - Pathfinding
  - Object navigation
  - Integrate with movement systems
- Match application themes
  - Adjust lighting settings
  - Adjust environmental settings
- Create ambiance
  - Dynamic weather
  - Environmental changes
  - Time-based changes
- In-app visual guidance
  - Maps
  - Navigational tools

1.5 Given a scenario, implement interactive features to guide and engage users throughout the prototype.

- User guidance
  - Tutorials
  - Tooltips
  - Guidance systems
- User tracking
  - Achievements
  - Milestone systems
  - Power-ups
  - Collectibles
- In-app capture systems
  - Audio
  - Photos
  - Screenshots
  - Videos
- Augmented reality (AR)

DOMAIN 2: Programming and Scripting (26%)

Candidates must be able to develop modular and maintainable C# scripts, implement object-oriented programming principles, and integrate advanced scripting techniques for Unity applications.

2.1 Given a scenario, develop C# scripts to implement core functionalities and interactions.

- Core functionalities
  - Input handling
  - Scene transitions
  - Audio management
  - UI navigation
  - Data persistence
- Character and object interactions
  - Buttons
  - Toggles
  - Triggers and colliders
  - Pickup and inventory systems
  - Custom events
- Control and logic flow
  - Conditionals
  - Loops
  - Switch statements
  - Coroutines
  - State management
- MonoBehaviour functions
  - Start()
  - Update()
  - FixedUpdate()
  - LastUpdate()
  - OnEnable()
  - OnDisable()
  - Awake()

2.2 Given a scenario, utilize essential data structures, event-driven design, and proven object-oriented programming patterns to create modular, maintainable, and high-performance C# code.

- Data structures
  - Arrays
  - Lists
  - Dictionaries
    - Keys
  - Queues
  - Stacks
  - HashSets
  - Custom types
    - Custom classes
    - Custom structures
  - Generic collections
    - Type-safe and reusable data storage
- Event-driven design
  - Events
  - Delegates
  - Decoupled systems
  - Observer pattern
- Error handling
  - Try-catch blocks
  - Defensive coding practices
  - Logging
  - Debugging strategies
- Object-oriented programming patterns
  - Singleton
  - Observer
  - Flyweight
  - Dependency injection
  - Encapsulation
- Modular coding principles
  - SOLID
    - Single responsibility principle (SRP)
    - Open/closed principles (OCP)
    - Liskov substitution principle (LSP)
    - Interface segregation principle (ISP)
    - Dependency inversion principle (DIP)
  - Readability
  - Reusability
  - Maintainability

2.3 Given a scenario, utilize ScriptableObjects, state machine patterns, and procedural generation techniques to manage data, drive AI behavior, and implement advanced game systems in Unity applications.

- ScriptableObjects
  - Manage reusable data and settings
  - Leverage ScriptableObjects for data management
  - Modularize game systems for reuse
    - Inventory
    - Skills
    - Level configurations
  - Create catalog systems for items, abilities, or levels
- Asset management
  - Organize and manage assets efficiently
  - Optimize asset usage and memory
  - Dynamically load and unload assets
- Artificial intelligence (AI)
  - Implement AI to interact/respond to user actions
  - Program adaptive AI reactions
  - Create state machines to manage AI behaviors
  - Use behavior trees for complex decision-making
- Script randomization systems
  - Spawn points
  - Loot tables
  - Procedural environments
- Procedural generation systems
  - Levels
  - Terrains
  - Assets
  - Natural environment creation using noise functions

2.4 Given a scenario, integrate advanced C# features to enhance code flexibility and performance, including asynchronous processes, reflection, coroutines, and dependency injection.

- Asynchronous operations
  - Async and await
  - IEnumerator and yield return
  - Asynchronous scene loading
- Coroutines in Unity
  - Time-based asynchronous tasks
  - Manage behavior over frames
- Input management
  - Input rebinding at runtime
  - Persistent user preferences
  - Save and load rebind configurations
- Reflection and advanced object manipulation
  - Access/modify at runtime
    - Properties
    - Methods
    - Fields
  - Custom attributes
    - Code markup
    - Metadata
    - Editor scripting
    - Exposing properties at runtime
  - Dynamic plugin loading
  - Runtime type discovery
  - Building modding interfaces
- Dependency injection (DI)
  - Decouple dependencies
    - Interfaces
    - Abstract classes
  - Use for flexibility
  - Use for testable code
  - Integrate DI frameworks

2.5 Given a scenario, create custom editor tools and inspector enhancements to streamline Unity workflows, while employing unit tests to maintain code quality.

- Unity Editor Tools
  - Develop custom tools to streamline workflows
    - Automate or simplify repetitive tasks
    - Create for player/user-generated content
  - Implement custom menus, windows, or panels
- Inspector Customization
  - Expose variables and settings through custom inspectors
  - Create custom Inspector Editors to enhance workflow and usability
- Unit Testing
  - Implement unit testing for C# scripts to ensure code quality
  - Test critical game systems for reliability and performance

2.6 Given a scenario, design and integrate a wide range of systems to build scalable and responsive Unity applications, including input handling, AI, data persistence, object pooling, and script interactions.

- Input Handling
  - Create input bindings for various devices
  - Manage input controls for dynamic gameplay
  - Write responsive touch controls
- AI and Non-player Systems
  - Script behaviors for AI or non-player entities
  - Script conversation and dialogue systems
- Object Management
  - Pooling techniques for efficient object creation and destruction
  - Handle dynamic object instantiation and destruction
  - Utilize GetComponent<T> to reference core components
- Data Persistence
  - Develop save/load systems for persistent data
  - Set up checkpoints, save points, or progress tracking systems
  - Serialize and deserialize data for storage
  - Write C# code to parse JSON, CSV, or XML
- Environmental Effects
  - Script environmental effects like wind, weather, and object movement
  - Implement time-based systems for animations or events
  - Handle collision and trigger events
- Script Interactions
  - Facilitate using public methods, events, and interfaces
  - Create event-driven programming patterns for systems
  - Work with events for game object lifecycle management
- Code Optimization
  - Use Linq for efficient data filtering and processing
  - Utilize collections and Linq for data management
  - Create reusable utility functions and helper classes
- Advanced Scripting
  - Implement custom data types with enums and flag
  - Apply C# attributes for editor-only functionality
  - Create singleton managers for global systems control
  - Implement exception handling to improve code reliability

DOMAIN 3: User Interface (UI) and User Experience (UX) (14%)

Candidates must be able to design intuitive user interfaces, implement responsive layouts, and enhance user experience through interactive UI elements and HUD components.

3.1 Given a scenario, construct cohesive and responsive user interfaces that adhere to a consistent theme by leveraging Unity's Canvas system and proper layout design.

- UI/UX Design
  - Design user interfaces (UI)
    - Status indicators
    - Progress bars
  - Design user experience (UX)
    - Intuitive UX
    - Consistent UX
  - Design a cohesive UI/UX
- Unity's Canvas system
  - Create and organize different views
  - Canvas modes
    - Overlay UI
    - Screen-space UI
    - World-space UI
- Create and adapt responsive UI layouts
  - Screen sizes and resolutions
  - Different devices and platforms

3.2 Given a scenario, develop interactive UI elements that respond dynamically to user gestures and event triggers.

- UI Interactions
  - Button clicks
  - Touch gestures
  - Event triggers
  - Drag-and-drop
- Menus and Navigation
  - Program menus
  - Navigation flows
  - UI transitions
  - Dynamic menus
  - Pop-ups
  - Customizable settings menus
- Dynamic UI Elements
  - Script elements responsive to user actions
  - Interactive icons for inventory or catalog systems
  - Custom UI controls for advanced interaction

3.3 Given a scenario, enhance user engagement and clarity through HUD elements, animated transitions, contextual tooltips, and optimized user interface (UI) effects.

- Create/implement HUD and data displays
  - HUD elements for real-time display
  - HUD elements for VR/AR experiences
  - Raycasting for object selection or line-of-sight mechanics
- Create/implement UI animations and transitions
  - UI transitions and animations
  - Animated tooltips and contextual help
  - Animate UI changes using tweening libraries
  - Create feedback mechanisms
    - UI transitions
    - Screen shakes
- Advanced UI Features
  - Adapt context-sensitive UI elements to user actions
  - Create responsive UI elements
  - Design and implement loading screens
  - Implement scrollable UI components
- UI Optimization
  - Implement caching for frequently used UI elements
  - Use Canvas Render Modes for different UI needs
  - Implement UI masking for clipping and visual effects

3.4 Given a scenario, design and implement localization, accessible features, and performance optimizations to handle device constraints and differing resolutions to ensure broad usability.

- Integrate locations in the UI to support multiple languages
- Create and implement accessibility features
  - High-contrast UI modes
  - Text-to-speech for UI elements
  - Keyboard and controller navigation support
- Implement responsive design
  - Support various screen resolutions
  - Support different aspect ratios
  - Adapt UI to various devices/formats
- Optimize UI performance
  - Object pooling
  - Caching
  - Reduce lag
  - Increase responsiveness

DOMAIN 4: Physics, Animation, and Effects (16%)

Candidates must be able to implement physics-based mechanics, create smooth animations, and develop particle-based visual effects for immersive game experiences.

4.1 Given a scenario, create realistic and performant physics systems using collision detection, rigidbodies, and movement mechanics that respond accurately to user input.

- Physics interactions
  - Object movement
  - Collision detection and handling
  - Rigidbodies
  - Colliders
- Dynamic object movement and forces
  - Forces
  - Torque
  - Jump
  - Lateral movement
  - Vertical movement
  - Joint physics
- Environment
  - Interactive
  - Destructible
- Optimization
  - Maintain performance
  - Maintain realism

4.2 Given a scenario, build and synchronize 2D and 3D animation systems using Animator Controllers, Inverse Kinematics (IK), and code-based triggers for fluid and reactive user experiences.

- Utilize Animator Controllers
- Animate 2D/3D objects
- Animate UI elements
- Utilize sprite sheets
- Sync animations with user actions
- Control animation states dynamically
- Transition between animation states
- Inverse Kinematics (IK)
  - Limb movement
  - Object manipulation
- Animation retargeting
  - Mecanim retargeting
- Implement various system behaviors

4.3 Given a scenario, design engaging particle-based visual effects to enrich user feedback, including collision-aware simulations, destructible objects, and post-processing enhancements.

- Configure particle effects
- Implement particle systems
- Add collision-aware particle effects
- Develop custom post-processing
- Enhance visuals with VFX
- Create water or fluid simulations
- Implement object destruction effects

4.4 Given a scenario, leverage advanced materials, shaders, and lighting techniques to achieve high-quality visuals and enhance depth and realism in Unity projects.

- Apply materials and shaders
- Create custom shaders
- Use Shader Graph for effects
- Create procedurally-generated visuals with shaders
- Configure lighting and shadows
- Use volumetric lighting for depth
- Implement post-processing lighting effects

4.5 Given a scenario, create cinematic experiences and advanced animations by utilizing Unity's Timeline, motion capture data, and facial animation.

- Utilize Unity's Timeline for cutscenes
- Integrate motion capture data in Timeline
- Implement facial animations in Timeline
- Configure physics for vehicles and machinery

4.6 Given a scenario, enhance user immersion with dynamic audio-visual events, sound effects, and music by synchronizing them through Unity's audio and scripting systems.

- Integrate audio events for interactivity
- Add sound effects and music with Unity's Audio System
- Manage audio using C# scripts
- Implement video playback with Unity's VideoPlayer component

DOMAIN 5: Optimization and Performance (14%)

Candidates must be able to optimize Unity applications by improving rendering performance, tuning physics parameters, and using profiling tools to identify and resolve performance bottlenecks.

5.1 Given a scenario, apply Unity's profiling tools to diagnose and resolve performance bottlenecks, memory leaks, and garbage collection inefficiencies.

- Use Unity Profiler to identify bottlenecks
- Debug and optimize code with profiling tools
- Profile and optimize memory usage
- Manage memory allocation
- Identify and resolve memory leaks
- Optimize performance for mobile devices
- Improve performance on lower-spec hardware

5.2 Given a scenario, streamline rendering performance by reducing unnecessary draw calls, implementing culling techniques, and employing lighting best practices.

- Rendering Optimization:
  - Reduce unnecessary draw calls
    - Optimize frame rates
    - Achieve efficient rendering
  - Implement object culling techniques
  - Minimize overdraw in UI and shaders
  - Optimize lighting with real-time solutions
  - Utilize baked lighting to improve performance

5.3 Given a scenario, improve real-time performance by tuning physics parameters, employing object pooling, and writing efficient, well-structured scripts.

- Physics Optimization
  - Adjust physics settings to meet application requirements
  - Optimize physics for different platforms (PC, mobile, console)
  - Implement object pooling for repeated elements
  - Optimize scripts and code structure
  - Cache component references
  - Use efficient data structures to reduce overhead

5.4 Given a scenario, organize and optimize assets, scenes, and build configurations to deliver efficient, streamlined deployments across multiple platforms.

- Configure scene lighting for performance and aesthetics
- Create and manage layers and tags for scene organization
- Use Asset Bundles for efficient content loading
- Implement Level of Detail (LOD)
- Compress and optimize texture assets
- Use asset compression to reduce application size
- Configure build settings to deploy on multiple platforms

5.5 Given a scenario, validate performance, functionality, and accessibility by conducting thorough testing and debugging across various platforms and hardware configurations.

- Testing for Usability and Performance
  - Real-time testing with Unity's Play Mode
  - Interaction mechanics for usability
  - Unit testing to ensure functionality
  - Unit testing to ensure stability
  - Cross-platform consistency testing
    - Input devices
    - Platforms
      - PC
      - Mobile
      - AR/VR
    - Responsiveness testing
      - Resolutions
      - Screen size
      - Platforms
    - Security testing
    - Connectivity testing
  - Conduct stress testing under load
  - Manage resource loading/unloading
  - Adjust prototypes based on feedback
- Debugging
  - Debug input handling
  - Debug interactions
  - Debug responsiveness
  - Debug events
  - Debug interactive AI objects
  - Remediate bugs/vulnerabilities
- Platform-specific adjustments
  - Configure graphics settings for device capabilities
  - Common pitfalls in mobile vs desktop vs console
    - Memory constraints
    - Build size constraints
    - Performance profiles
  - Set up monetization features
    - Advertisements
    - In-app purchases
    - Subscriptions
    - Licensing
- Validate accessibility features
  - Colorblind support
  - Control remapping

DOMAIN 6: DevSecOps (14%)

Candidates must be able to implement CI/CD pipelines, manage source control, integrate cloud-based services, and apply security best practices in Unity game development.

6.1 Given a scenario, maintain project cohesion by organizing assets, adhering to design documents, and optimizing models and textures for diverse target platforms.

- Follow design documents to align with project vision
- Manage and structure project assets
  - Scripts
  - Prefabs
  - Textures
- Create and manage prefabs for reusable objects
- Use interfaces to define reusable behaviors
- Import 3D models and configure them for Unity
- Configure and manage texture maps for realism
  - Normal maps
  - Bump maps
- Optimize textures and assets for performance and mobile compatibility

6.2 Given a scenario, facilitate seamless teamwork and code integrity through version control, task management tools, and collaborative reviews.

- Use Git for source control in Unity
- Resolve merge conflicts in Unity projects
- Integrate assets with designers and artists
- Collaborate with QA testers to identify bugs
- Utilize project management tools for task tracking
- Participate in code reviews

6.3 Given a scenario, establish continuous integration and continuous deployment (CI/CD) pipelines to streamline updates, patches, and third-party integrations.

- Implement CI/CD pipelines for projects
- CI/CD platforms
  - GitHub Actions
  - GitHub CI
  - Azure Pipelines
  - Jenkins
- Configure CI/CD pipelines for Unity-specific tasks
  - Command-line builds
  - Automated testing
  - Artifact generation for multiple platforms
    - PC
    - Mobile
    - Console
- Regularly push updates and patches to repositories
- Incorporate versioning systems
- Integrate third-party assets and plugins to extend functionality
  - Unity Package Manager
  - Conduct compatibility checks
  - Monitor dependencies

6.4 Given a scenario, incorporate networking, analytics, and communication features to support multiplayer, user tracking, and connected companion apps.

- Networking
  - Implement real-time multiplayer features
  - Implement real-time data exchange
  - Set up in-app voice or chat for communication
  - Use Unity Analytics to gather user data and behavior
  - Implement leaderboards
  - Implement user tracking
  - Integrate achievements and milestones
  - Develop companion apps for mobile integration

6.5 Given a scenario, leverage various cloud platforms and RESTful APIs to handle data storage, authentication, and dynamic content updates within Unity applications.

- Cloud functionality and data handling
  - Use UnityWebRequest for HTTP requests
  - Utilize JSON formatted data
  - Utilize RESTful APIs
    - CRUD (Create, Read, Update, Delete)
  - Perform API callbacks
  - Conduct asynchronous data retrieval
  - Update game content dynamically from the cloud
  - Serialize and deserialize data for cloud storage
  - Implement cloud-based save/load systems for cross-platform functionality
- AWS Integration
  - Utilize the AWS SDK for Unity
  - Utilize DynamoDB for data storage
  - Authenticate with AWS Cognito
  - Upload/download files in S3
  - Utilize AWS Lambda for backend logic
  - Monitor Unity logs with AWS CloudWatch
  - Configure IAM roles for Unity API access
- Azure Integration
  - Connect to Azure Cosmos DB
  - Authenticate with Azure Active Directory
  - Send and retrieve data from Azure Table Storage
- Google Cloud Integration
  - Utilize the Google Cloud SDK for Unity
  - Connect to Firebase Realtime Database
  - Authenticate users with Firebase Authentication
  - Store and retrieve data from Google Cloud's Firestore