Solan Games Test Assignment

Mikhail Oleksenko

KEVURU GAMES

Art Asset Preparation

1.1 Pixel Art Guidelines

- **Consistent Resolution**: Ensure all pixel art assets adhere to a consistent resolution. Define a pixel-per-unit (PPU) setting that matches the intended scale of your sprites in the game world.
- **Palette Limitation**: Use a limited color palette to maintain a cohesive style. Agree on a palette early in the project.
- **Grid Alignment**: Keep all sprites aligned to a pixel grid to avoid sub-pixel rendering issues, which can cause blurriness.
- **Outline and Shading**: Use consistent outline thickness and shading styles. Avoid gradients and instead use dithering or selective pixel placement.

1.2 Cartoon Style Guidelines

- **Vector Art**: If using vector art, ensure that assets are properly converted to raster images at the correct resolution.
- **Smooth Lines**: Maintain smooth and clean lines. Use anti-aliasing judiciously to ensure compatibility with pixel art.
- **Consistent Style**: Maintain consistent proportions, line thickness, and shading techniques across all cartoon-style assets.

Import Settings

2.1 Sprite Import Settings

- **Pixels Per Unit (PPU)**: Set the PPU to ensure the sprite's size matches the intended in-game size.
- **Filter Mode**: Use "Point (no filter)" for pixel art to maintain crisp edges. For cartoon art, "Bilinear" might be acceptable if it doesn't interfere with the pixel aesthetic.
- **Compression**: Disable compression for pixel art to avoid artifacts. Use lossless compression if necessary for cartoon art.
- **Sprite Mode**: For animations, set the sprite mode to "Multiple" and slice the sprite sheet appropriately.

Animation

3.1 Animation Tools

- **Unity Animator**: Use Unity's Animator for character animations. Ensure transitions between animations are smooth and logical.
- **Frame Rate**: Keep the frame rate consistent across animations to avoid jittery movement. For pixel art, common frame rates are 8, 12, or 24 FPS.

- **Animation Curves**: Use animation curves for smoother transitions and movements, particularly for cartoon elements.

Level Design

4.1 Tilemaps

- **Tilemap Grid**: Use Unity's Tilemap system to create levels. Ensure that tiles align perfectly with the grid to prevent visual seams.
- **Colliders**: Use Tilemap Colliders for efficient collision detection. Customize colliders for interactive elements.
- **Layering**: Organize tiles into layers (background, foreground, interactive objects) to maintain visual clarity and depth.

4.2 Parallax Scrolling

- **Layering**: Implement parallax scrolling by moving background layers at different speeds relative to the camera. This adds depth without affecting the pixel-perfect nature of foreground elements.
- **Pixel Perfection**: Ensure that parallax layers maintain pixel-perfect movement by adjusting their speed based on the camera's movement.

Technical Implementation

5.1 Camera Setup

- **Orthographic Camera**: Use an orthographic camera for a consistent 2D view.
- **Pixel Perfect Camera**: Use Unity's Pixel Perfect Camera component to ensure pixel art remains crisp and aligned with the pixel grid.
- **Resolution and Aspect Ratio**: Design for a target resolution and aspect ratio. Use letterboxing or pillarboxing to handle different screen sizes.

5.2 Performance Optimization

- **Sprite Atlases**: Use Sprite Atlases to reduce draw calls and improve performance.
- **Object Pooling**: Implement object pooling for frequently instantiated objects to reduce garbage collection overhead.
- **Batching**: Take advantage of Unity's Sprite Renderer batching by minimizing the number of materials and textures used.

Quality Assurance

6.1 Testing

- **Playtesting**: Regularly playtest to ensure the combination of styles looks cohesive and that gameplay elements function as expected.
- **Visual Consistency**: Continuously review visual elements for consistency in style, alignment, and resolution.

6.2 Debugging

- **Logs and Profiling**: Use Unity’s Profiler to monitor performance and identify bottlenecks. Use logging to track and resolve bugs.

Collaboration

7.1 Version Control

- **Git**: Use Git for version control. Establish branching strategies to manage features and bug fixes.
- **Asset Serialization**: Set asset serialization mode to “Force Text” to make diffs more manageable.

7.2 Documentation

- **Technical Documentation**: Maintain comprehensive documentation of art guidelines, technical setups, and development workflows.
- **Communication**: Use project management and communication tools (e.g., Trello, Slack) to coordinate tasks and share progress.