Escape
Report

Contents

1 Idea

The original idea for the game was to implement a ‘shifting’ mechanic, each level would be built up of multiple layers, some of which would be out of phase of the player, appear translucent and red/blue shifted according to how out of phase they were with the player. However this was causing complications during development and creating engaging puzzles would’ve taken much more time (including requiring more play testing by those unaware of the puzzle’s solutions). The shifting phase idea was then scrapped and instead the game is being built towards a generic escape/puzzle platformer.

2 Code Structure

2.1 Objects

Object is the base class for all objects that appear in game. This class is specialised for individual needs, such as *Player*, *Platform* and and *Exit*.

Classes can also inherit from a pair of linked abstract classes, *Trigger* and *Actuator*. These classes allow objects to be connected and act as switches or be switchable.

2.1.1 Trigger

Anything that is a *Trigger* can be used (default key: 'E'). *Trigger*s are linked to *Actuator*s and *actuate* the *Actuator* when triggered. Classes implementing *Trigger* must implement $boolon()const$ and $voidset()$.

2.1.2 Actuator

Anything that is an *Actuator* can be activated by using it's linked *Trigger*. Classes implementing *Actuator* must implement $voidactuate()$.

2.2 Visages

Visage is the base class for anything that can appear within the game that's not part of the user interface. The Visage class is subclassed into a few classes to provide different visages.

2.2.1 VisagePolygon

A *VisagePolygon* represents a simple single colour polygon. Functions exist to create rectangles, triangles and regular N-sided polygons.

2.2.2 VisageTexture

A *VisageTexture* represents a texture rendered on a quad. Textures can be a sprite sheet. Textures can be scrolled and repeated on the quad, making it suitable for things like scrolling backgrounds.

2.2.3 VisageText

A *VisageText* can draw text in the game world. Fonts are baked on load and text is rendered on quads. Any font can be used, provided a ttf file exists for it in the 'font/' directory.

2.2.4 VisageComplex

A *VisageComplex* can hold multiple *Visage*s, allowing complex composite visages containing multiple graphics, e.g., combining a VisageTexture and ParticleSystem as one visage.

2.2.5 ParticleSystem

A *ParticleSystem* creates and manages particles, which can be used for creating smoke, fire, fog and similar effects. The particles can have random lifespans and can have various properties such as size, colour and speed. Properties are interpolated from the start to end of life.

2.2.6 Animatrix

An *Animatrix* is not a subclass of *Visage*, but a *Visage* can have any number of Animatrices. An *Animatrix* is used to animate a visage to perform looped size, colour, rotation, and positional changes. Changes are interpolated and multiple animatrices can be given, allowing complex routines to be built which occur one after another. Colour modifications are multiplicative and the interpolation is in HSV colour space giving better transitions.

2.3 Scene graph

Objects are arranged in a *SceneGraph* class to ensure things are rendered in the correct order. Since everything is on the Z=0 plane, the scene graph is separated into multiple layers, instead of ordering objects by their depth.

2.4 Graphical User Interface

The graphical user interface is built up of a few classes that represent different elements; *GUIElement* is the base class, which is extended to *GUIWindow*, *GUILabel* and *GUIImage*. All GUI elements can be placed in terms of percentages of the size of their parent and in absolute positions (or combined), allowing quick and powerful UI design.

2.4.1 GUI Event Handling

Instead of defining events, such as 'ON_CLICK' and the likes, the GUI system exposes a method of registering callbacks as events and the callback of the event. This allows easy custom events and actions to happen from those events, including having an empty event to run a callback every frame.

3 Testing

Most testing has been performed ad-hoc, without use of a testing framework.

4 AI

At time of writing, there are three types of AI, the *Follower*, the *Camera* and the *Turret*.

4.1 Follower

The *Follower* follows the player. Aiming for a spot near the player's head it accelerates towards it. The acceleration increases as the distance increases. While idling this gives a nice result of the follower orbiting the player. The follower does not care for the geometry of the level and will freely clip through any solid surfaces though.

4.2 Camera

The *Camera* is a mostly non-functional AI and is purely for visual purposes. It remains stationary, when the player is in line of sight of the camera it'll rotate the aim at the player. While the player is visible it shows with a small red dot, like the camera is active.

4.3 Turret

The *Turret* is a specialisation of the *Camera*. Like the *Camera*, it turns to face the player when the player is in line of sight. When the angle to the player from the facing direction of the turret is small enough the turret begins to fire small, high rate of fire projectiles in the direction it's facing with a little bit of spray.

5 Window creation & handling

Window creation and handling is handled by 'freeGLUT'. This was originally chosen as it was cross platform and free software, allowing the game to run on Linux based operating systems and Windows. In time with development issues have arisen though, with time spare or in future occasions a switch to GLFW would be made as it seems to be superior to freeGLUT.