Technical Design Document – PurrEmotion

# Title Page

PurrEmotion

# Document History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author(s) | Changes |
| 1.0 | 04/14/2024 | Herbert Cunha |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Table of Contents

# Game Summary

PurrEmotion is a heartwarming adventure game where players take on the role of a wise and caring cat navigating through a beautifully crafted environment. The game centers around the cat's mission to alleviate its owner's struggles with stress, anxiety, and depression. Through intuitive gameplay, players interact with various elements within the household, each interaction aimed at bringing comfort, joy, and relief to the cat's human companion.

# Development Environment

## Development Hardware

Windows OS, 16GB RAM, 2 Monitors, Intel Core I5 Notebook.

## Programming Languages

C++

## Development Tools

Visual Studio, GitHub, DALL-E

## External Code

SFML - https://www.sfml-dev.org/

## Game Engine

Game Engine that we made together during the course classes.

# Architectural Analysis

## Classes

Describe the classes that will have to be implemented. For each class, provide:

* Its responsibilities
* How it collaborates with other classes

|  |  |  |
| --- | --- | --- |
| Class | Responsibilities | Collaborations |
| Command.h | • Encapsulate action details within the game, providing a uniform approach to handle user inputs or game actions. • Store identifiers such as the name and type of the command, which can be used to determine the action to be executed. | GameEngine or Controller Classes Event Handling System |
| Components.h | The Component system in the game architecture utilizes a classic entity-component design pattern to promote modularity and flexibility in game object behavior and appearance. Each component represents a distinct aspect of functionality that can be attached to game entities. | CAnimation CSprite CTransform CBoundingBox CState CInput CPowerUps CScript |
| Entity.h | • Represent individual objects within the game world, each possessing various behaviors and properties as defined by their components. • Manage lifecycle events of the entity such as creation, activation, deactivation, and destruction. | EntityManager Components (CSprite, CAnimation, etc.) |
| EntityManager.h | • Manage the lifecycle of all Entity instances within the game. • Organize entities by tags for efficient categorization and retrieval. | Entity EntityVec and EntityMap |
| GameEngine.h | • Centralize and manage the core game loop, including initialization, rendering, and updates. • Handle scene transitions, maintaining a map of all available game scenes and managing the active scene | Scene Assets sf::RenderWindow sf::Text and sf::Time |
| MusicPlayer.h | • Manage background music for the game. • Control playback of music tracks, including play, stop, and pause functionalities. • Adjust music volume and manage different music tracks loaded from files | sf::Music |
| Physics.h | • Provide core physics calculations related to entity interactions within the game, specifically focusing on collision detection. • Determine the overlapping area between two entities, which is crucial for resolving collisions and interactions in gameplay. | Entity sf::Vector2f |
| Scene.h | • Serve as the base class for different scenes in the game, such as menus, gameplay levels, and game over screens. • Manage entities within the scene using | GameEngine EntityManager Command |
| Animation.h | Manage and control frame-by-frame animation for sprites.  Handle the timing and sequence of frames within an animation cycle.  Control the playback of the animation (start, stop, and determine if it has ended).  Provide access to the associated sprite and its bounding box dimensions | sf::Sprite sf::Texture sf::Time |
| Assets.h | • Manage and centralize access to game assets such as fonts, textures, sound effects, animations, and sprite definitions. • Load assets from files based on configuration or specific paths. | sf::Font sf::Texture sf::SoundBuffer Animation sf::Sprite |
| Scene\_PurrEmotion.h | • Manage gameplay mechanics specific to the game mode, including player behavior, scene dynamics, and game rules. • Control movement, animations, collisions, and entity spawning within the game. | EntityManager GameEngine Entity Scene |
| Scene\_Menu.h | • Manage the main menu interface of the game, allowing players to navigate through options such as starting the game, viewing credits, or exiting the game. • Handle the graphical presentation of the menu, including background animations and text display | GameEngine Scene sf::Text, sf::Font, sf::Sprite, sf::Texture |
| SoundPlayer.h | • Manage playback of sound effects within the game, including starting, stopping, and positioning sounds based on game events and interactions. • Maintain a pool of sound objects to ensure efficient sound management and playback. | sf::Sound sf::SoundBuffer sf::Vector2f |

Present class diagrams that show the relationships between classes. Show only the most essential attributes and methods for each class.

## Game Loop

Describe, in order, the sequence of activities that happen during each game loop. You must document this even if you’ll be using the “Clown Cannon” game engine.

# Technical Risks

List all technical risks that could make it difficult or impossible to complete the game. Examples:

* Uncertainty on how to implement a certain feature
* Uncertainty on if a certain feature can be executed fast enough in real time
* First time using a certain library

|  |  |  |
| --- | --- | --- |
| Risk | Severity | Mitigation (what is to be done to eliminate or minimize this risk) |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |