PACMANV2

State Manager

I decided to rewrite the state manager due to copyright concerns with the previous version. The new state manager is now a template class, allowing it to be inherited and modified to implement custom states. Making it a template helps avoid duplicating code for game states and entity states.

Two state type interfaces have been created: `IGameState` and `IPlayerState`. The `GameStateManager` class implements additional methods from its parent class, `StateManager`. A global pointer to `GameStateManager` is used across states.

Both Pacman and Ghosts inherit from `StateManager`. The `this` pointer is used to switch between states within the current entity state.

- Fixed the lifetime of the state manager, eliminating memory leaks.

Ghost Setup State

The setup state is implemented by assigning each ghost a target of a custom cell specified by the number 7. This approach eliminates the need for a predefined set of instructions for each map, allowing the ghosts to escape more dynamically.

Once a ghost reaches the target cell, it changes state and enters the chasing state, adjusting its target accordingly.

Observer Pattern

The Observer pattern is utilized in the `ScoreManager` and `EntityView` classes to handle changes and updates efficiently:

- `ScoreManager`: Acts as an observer that monitors changes in the game state, specifically events related to Pacman. It listens to notifications (or "events")

from a subject, such as when Pacman eats an item or dies. Based on these notifications, `ScoreManager` updates the game score accordingly.

- **`EntityView`**: Also uses the Observer pattern to respond to changes. It observes the `EntityModel` and listens for "Draw" events. When such an event is emitted, `EntityView` updates the display by drawing the entities (e.g., Pacman, ghosts) on the screen according to their current state and position.

Animations

The `Animation` class manages sprite animations by storing frames for each animation type in a map. It updates the current frame based on elapsed time and a specified switch interval, allowing for smooth transitions between frames. The `Start` and `Stop` methods control whether the animation is playing, while `Reset` sets the animation back to the beginning. The `Update` method advances the frame if the animation is running and the switch time has elapsed. The `GetCurrentFrame` method retrieves the frame currently being displayed.

Banana Entities

When Pacman eats a coin, the position of the coin is added to an array of available free positions in the world. Every 10 seconds, the world checks this array for available positions and uses a random number generator to select a position to spawn a banana. When Pacman eats a banana, he pauses for 1 second, and the animation stops.

Dynamic Maps

MapManager is implemented as a Singleton class, ensuring that only a single instance exists across the entire application. This central instance handles all map-related operations. The `LoadMaps` method loads map data from a JSON file, converting each entry into a `Map` object and storing them in a vector. The `NextMap` method advances to the next map in the list, looping back to the start when the end is reached, allowing for continuous map cycling. The `GetCurrentMap` method provides access to the currently selected map based on an internal index. This

design allows for seamless progression through maps as levels advance, ensuring circular gameplay through all available maps. Additionally, the dynamic creation of menu buttons for map selection means that if the number of maps changes, the menu logic automatically adapts, eliminating the need for manual updates each time a new map is added.

World & Player States

The world manages all entities within the game. It is initialized when a `LevelState` is created and destroyed when the level ends (`LevelExit`). The current level information is stored in the `ScoreManager`.

The world handles collisions between entities and is responsible for spawning or removing entities. However, the movement physics of each entity are handled individually by the entities themselves, through their respective states. To update each entity, the world simply calls `entity->Update`

Game Class

The `Game` class runs the main loop and calculates `deltaTime` for each tick, maintaining a fixed frame rate of 60 FPS. Within the `Run` method, it calls `HandleInput`, `Update`, and `Draw` methods from the `GameStateManager`. The `GameStateManager` then delegates the game logic to the current game state, ensuring that the game logic is managed by the active state.

Extra Features

A sound manager has been added to this project. It plays a background music and listen to event emitted by pacman to play additional sounds.