Pac-Man Project Report

https://github.com/VanNoten/Pac-Man

Chakkarin Van Noten, 20240497

2025-2026

Project Defence: Pacman Project.mp4 - Google Drive

Entity creation and observer attachment

The Abstract Factory pattern is used to connect the logic and application layers. When the World class creates an entity through the factory interface, the Concrete Factory (SFMLFactory) constructs the entity model from the logic layer, creates the corresponding view for it and attaches this view to the entity model. For the Pac-Man, Coin and Fruit entity models an additional score observer is attached to handle scoring events such as eating ghosts, coins and fruits.

Entity views hold a const reference to their corresponding entity model. This makes sure that views don't own the model and have read-only access to the model, which enforces MVC separation and prevents the view from affecting game logic. Using a reference rather than a pointer also guarantees that a view cannot exist without a valid model so null checks are not needed.

Movement System

The movement system combines continuous movement with tile-based direction changes. Entities move smoothly by using the formula position += speed * deltaTime. Direction changes are constrained to happen only when an entity is close enough to a tile center and wall collision detection is performed by checking whether the target tile in the direction of movement is a wall.

The decision to use tile-based direction changes was made because letting entities change direction at any moment caused them to cut corners and accidentally collide with walls. Performance was also worse due to bounding-box collision checks. The tile-based approach improved wall collision detection efficiency and gameplay quality.

For Pac-Man, current direction and wanted direction are stored. Player input immediately updates the wanted direction but the current direction only changes when Pac-Man is close enough to a tile center and the target tile is not a wall. An exception was made to reverse the current direction which happens instantly. This exception was necessary because requiring players to wait until a tile center made the controls feel unresponsive.

Ghost entities keep track of the last tile they made a direction decision on, which prevents them from making multiple direction decisions on the same tile.

Ghost AI Behavior

The ghost AI implementation required handling of some edge cases that were not explicitly explained in the project requirements. The locked ghost first determines whether a direction change should happen (50% chance). If direction should change, the current direction is filtered out of the available options before selecting a random direction, ensuring the ghost changes direction. If

direction shouldn't change but the current direction is blocked (e.g. at a corner) the ghost falls back to a random direction from the available options instead of leaving the ghost stationary.

When a ghost gets feared by Pac-Man eating a fruit, it immediately tries to reverse its direction. If the reverse direction leads the ghost into a wall, the ghost keeps its current direction but still enters feared mode.

Coordinate system

Entity coordinates in the world represent the center of the entity and not the corner, this simplifies the movement system. The world calculates a cell size as 2 / max(rows, columns), ensuring the entire map fits in the normalized [1, -1] coordinate space.

The camera class recalculates its projection variables which enables the game to respond to window resizing. The scale factor is determined by the minimum of the width and height of the window, preventing visual distortion.

Observer events

Entities notify observers when their state changes, this approach means views only update the sprite that needs to be shown when the state has changed instead of polling the state each frame.

The Move event, sent by Pac-Man and ghosts, is used by views to update their rendering position. The DirectionChanged event triggers sprite updates for Pac-Man and ghosts so they have the correct sprite for the current direction they're facing.

CoinCollected, FruitCollected and GhostEaten events are observed by the score class to increase the score. Additionally, the CoinCollected, FruitCollected events are also observed by their views. If a coin or fruit has been collected their views will not render them anymore.

World events like LevelCleared and GameOver trigger a score increase and saving the current score to disk.

State management

The StateManager has three operations: changeState (which clears the stack and pushes a new state), pushState and popState. This stack-based design enables the ability to pause the game by pushing PausedState on top of PlayingState and resuming the game by popping it.

Level transitions are handled by creating a new PlayingState instead of resetting an existing one. The current score, remaining lives and incremented level are passed to the constructor.

States store a reference to StateManager instead of a pointer, ensuring valid access without the need for null checks.

Scaling difficulty

Ghost speed gets incremented by a certain amount each time a level is cleared but is capped at a maximum value to prevent the game from becoming impossible to play. Feared mode duration decreases each time a level is cleared and there is a minimum duration, so players always have some time to eat ghosts.

Rendering and animation

Entity views contain a z-level which is used to control the rendering order. Walls, coins and fruits have a z-level of zero, while Pac-Man and ghosts have a z-level of one. The PlayingState retrieves the views from the SFMLFactory after world loads the map. Those views are then sorted by z-level to ensure that walls, coins and fruits are rendered first, and Pac-Man and ghosts are rendered on top of them. Without this z-level system, coins and fruits would appear above ghosts (not Pac-Man because it eats them).

Sprite animations are managed by each view using timers. When the accumulated time is larger than the animation speed, the current frame changes and the timer resets. Sprite rectangles are calculated in the constructor of the view instead of calculating them every frame. This improves performance of the game.

Documentation

Core classes and interfaces are documented using Doxygen-style comments. Allowing the full API documentation to be generated automatically using Doxygen.

Design patterns used:

Abstract Factory: AbstractFactory, SFMLFactory

Observer: EntityView, Score

Subject: EntityModel, World

State: MenuState, PlayingState, PausedState, VictoryState, GameOverState

Singleton: Stopwatch, Random, ResourceLoader

MVC: Models in logic/Entities, Views in application/Views, World as controller