# COMP 504 Graduate Object-Oriented Programming and Design

# PacMan App API Specification

Group Apex
Rice University
Department of Computer Science

Nov 28th, 2023

Team Lead: Hongming Zhang
Tech Lead: Sung-Yi Wang
Developer: Jiaqi He
Developer: Juncheng Zhou
Documentation: Feifei Li

# Table of Contents

# Introduction

This document outlines the API specification for our online PacMan game application. This game app is designed by using Javascript and Java Spark, The game is designed with multiple components, focusing on user interaction, game mechanics, and Ghost behaviors.

Pac-Man is an action maze chase video game; the player controls the character through an enclosed maze. The objective of the game is to eat all of the dots placed in the maze while avoiding four colored ghosts. When Pac-Man eats all of the dots, the player advances to the next level.

**Key Features**

- Difficulty Levels: At the start of the game, players can choose from various difficulty levels, offering a tailored challenge to both new and experienced gamers.

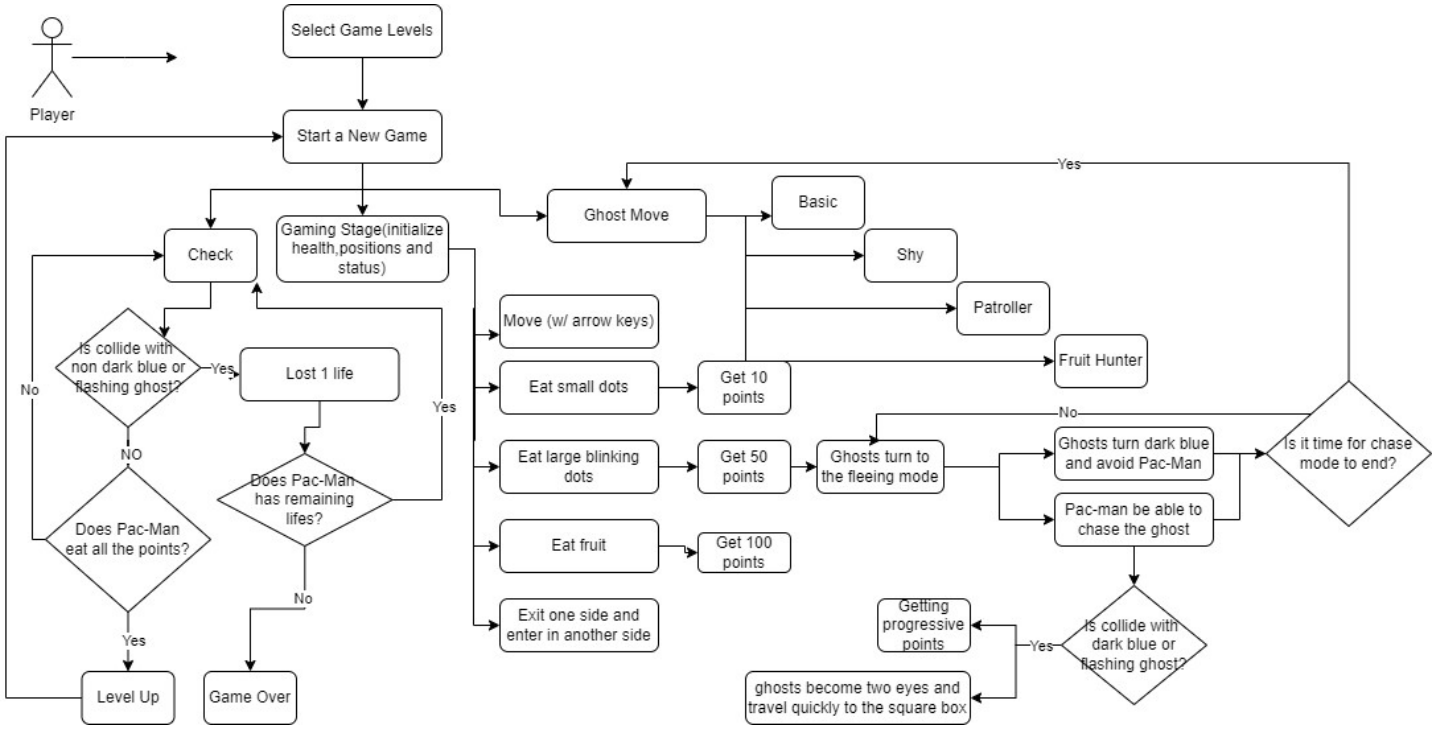
- User Interaction: Players control PacMan using the arrow keys, navigating through the maze to collect points and avoid ghosts.

- Scoring System: As PacMan eats dots and power-ups, players earn credits, adding an extra layer of incentive and competition.

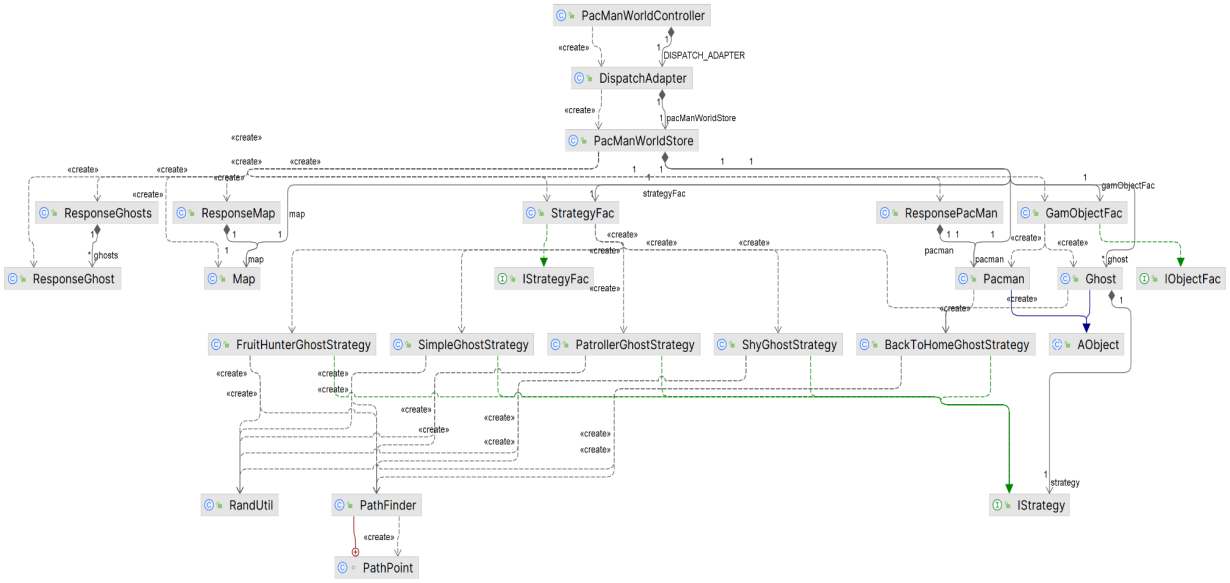
- Ghost Behaviors: Utilizing distinct AI strategies, ghosts will move towards or away from PacMan, each employing unique patterns to challenge the player. This variety ensures each game experience is dynamic and unpredictable.

- Level Progression: Players advance to the next level after clearing all dots from the maze, with the game increasing in difficulty. Pac-Man must complete these levels without losing three lives to sustain the game's progression.

# Use Cases




# UML Diagram



# Interface and abstract class

## DispatchAdapter

The DispatchAdapter class serves as an adapter between the view (paint objects) and the controller in a PacMan game. It provides functions for initializing the game, updating PacMan, the map, and the ghosts, and setting the game level. The class also implements the Singleton pattern to ensure that there is only one instance of the DispatchAdapter throughout the application.

| Function Header | Description |
|---|---|
| public DispatchAdapter() | Constructor for the DispatchAdapter class. |
| public static DispatchAdapter makeDispatchAdapter() | Singleton constructor for DispatchAdapter. |
| public static ResponsePacMan updatePacMan(String operation) | Operates the Pacman and returns a response. |
| public static void init() | Initializes the game. |
| public ResponseMap update() | Updates the map and returns a response. |
| public ResponseGhosts updateGhosts() | Updates the ghosts and returns a response. |
| public void setLevel(int level) | Sets the game level. |

## Map

The Map class represents the game map for a PacMan game. It contains a 2D array representing the layout of the map with various elements, such as walls ('1'), empty cells ('0'), dots ('-1'), PacMan ('2'), and fruit ('3'). The class provides functions to interact with the map, including adding score based on PacMan's position, setting and getting the map, generating fruit, and managing the game score. Additionally, it has a constructor that initializes the map by replacing some '0' points with '-1' points to create the initial dots.

| Function Header | Description |
|---|---|

| | |
|---|---|
| public Map() | Constructor for the Map class. |
| public int[][] getMap() | Get the current map. |
| public boolean addScore(int x, int y) | Adds score based on the content of the map cell at position (x, y). Returns true if a ghost is eaten. |
| public int[][] setMap(int[][] map) | Sets the map to a given 2D array and returns the updated map. |
| public void generateFruit() | Randomly generates a fruit (represented by '3') in the map. |
| private static void replaceZeroPointsWithMinusOne(int[][] map, int n) | Replaces '0' points with '-1' points (dots) in the map, randomly choosing 'n' points. |
| public int getScore() | Get the current score. |
| public void setScore(int score) | Set the score to a specified value. |

## AObject

The AObject class is an abstract class representing objects in a game. It provides basic attributes and methods that are common to all game objects, such as position (x, y) and direction. Subclasses can inherit from AObject to define specific game objects with additional attributes and behavior.

| Function Header | Description |
|---|---|
| public AObject(int x, int y, String direction) | Constructor for the AObject class, initializes the object's position (x, y), and direction. |
| public int getX() | Get the x coordinate of the object. |
| public int getY() | Get the y coordinate of the object. |
| public String getDirection() | Get the direction of the object. |
| public void setX(int x) | Set the x coordinate of the object. |
| public void setY(int y) | Set the y coordinate of the object. |
| public void setDirection(String direction) | Set the direction of the object. |

## GamObjectFac

The GamObjectFac class is responsible for encapsulating the logic for creating Pacman and Ghost objects with the desired characteristics. It implements the factory pattern by providing methods for creating these objects, which can be used to instantiate game objects with different properties as needed in your game application.

| Function Header | Description |
| --- | --- |
| public Pacman makePacman(int x, int y, String direction) | Creates a Pacman object with the specified position (x, y) and direction. |
| public Ghost makeGhost(int x, int y, String direction, String color, IStrategy strategy) | Creates a Ghost object with the specified position (x, y), direction, color, and strategy. |

The IObjectFac interface follows the factory pattern, providing a common interface for creating game objects while allowing different implementations (such as the GamObjectFac class) to handle the actual object creation logic with specific attributes.

## Ghost

The Ghost class is a fundamental component of the PacMan game, responsible for handling the movement, collision detection, and behavior of ghosts in the game world. It interacts with the game map (Map) and Pacman (Pacman) to implement the game's dynamics.

| Function Header | Description |
| --- | --- |
| public Ghost(int x, int y, String direction, String color, IStrategy strategy) | Constructor to create a Ghost object with specific attributes like position, direction, color, and strategy. |
| public String getColor() | Get the color of the ghost. |
| public void setColor(String color) | Set the color of the ghost. |
| public int getLevel() | Get the level of the ghost. |
| public void setLevel(int level) | Set the level of the ghost. |

| | |
|---|---|
| public IStrategy getStrategy() | Get the current strategy of the ghost. |
| public void setStrategy(IStrategy strategy) | Set the strategy for the ghost. |
| public IStrategy getPrevStrategy() | Get the previous strategy of the ghost. |
| public void setPrevStrategy(IStrategy prevStrategy) | Set the previous strategy of the ghost. |
| public int getVelocity() | Get the velocity of the ghost. |
| public void setVelocity(int velocity) | Set the velocity of the ghost. |
| public boolean getCanEaten() | Check if the ghost can be eaten. |
| public void setCanEaten(boolean canEaten) | Set whether the ghost can be eaten. |
| public boolean getBeEaten() | Check if the ghost has been eaten. |
| public void setBeEaten(boolean beEaten) | Set whether the ghost has been eaten. |
| public void move() | Move the ghost based on its direction and velocity. |
| public void ghostCollisionDetection(Ghost ghost, Map map, Pacman pacman) | Check for collisions between the ghost and the game map, as well as collisions with Pacman. |
| public void update(Ghost ghost, Map map, Pacman pacman) | Update the ghost's movement and behavior based on its strategy and other factors. |

## Pacman

The Pacman class represents a Pacman object in a PacMan game. It extends the abstract class AObject and has various attributes and methods to manage the behavior and movement of Pacman.

| Function Header | Description |
|---|---|
| public Pacman(int x, int y, String direction) | Constructor to create a Pacman object with specific attributes like position and direction. |

| | |
|---|---|
| public void move() | Move Pacman based on its current direction. |
| public int getLife() | Get the remaining life count of Pacman. |
| public void setLife(int life) | Set the remaining life count of Pacman. |
| public int getAteGhost() | Get the number of ghosts Pacman has eaten. |
| public void setAteGhost(int ateGhost) | Set the number of ghosts Pacman has eaten. |
| public int getEatenTime() | Get the remaining time for Pacman to be invulnerable after eating a ghost. |
| public void setEatenTime(int eatenTime) | Set the remaining time for Pacman's invulnerability. |
| public boolean pacmanCollisionDetection(String operation, Pacman pacman, Map map, ArrayList<Ghost> ghost) | Check if Pacman can move to the next position based on the current operation (up, down, left, right). |
| public void move() | Move Pacman based on its current direction. |

## Strategy

The IStrategyFac interface defines a factory for creating ghost strategies in a PacMan game. It includes a single method:

The StrategyFac class implements the IStrategyFac interface, providing a factory for creating different ghost strategies based on the strategy name.

# Design Patterns

## 1. MVC Design Pattern

Our application follows the MVC design pattern. We separate the application into three components: Model (data), View (user interface), and Controller (logic). This separation enhances code organization and maintainability.

## 2. Factory Design Pattern

The PacMan and Ghosts objects are constructed using the factory design pattern without a specific object class being specified. This abstraction makes object creation easier and encourages code reuse. The production of objects is captured by the Factory pattern. It offers a uniform interface for constructing things and abstracts away the specifics. This abstraction aids in shielding the client code from the complexity of object generation. We may simply alter the class of objects being generated by using a Factory without changing the client code. Additionally, it sets up object creation logic in an organized way.

## 3. Strategy Design Pattern

In our PacMan Game, the Strategy design pattern is used to develop several ghost strategies. We decided to manage these various strategies kinds using the Strategy pattern, which enables us to dynamically encapsulate and switch between strategies. Each message strategy has a particular rendering or processing algorithm. Because the algorithmic intricacies are kept outside from the primary game application logic by this encapsulation, the resulting code is clearer and more modular. The application implements various ghost strategies under the strategy design pattern, enabling the easy addition of new ghost behaviors and making the AI more adaptable.

## 4. Singleton Design Pattern

We used the Singleton design pattern in the PacManWorldStore. Throughout the duration of the program, the Singleton design makes sure that these stores only have one instance. The Singleton design guarantees that user data comes from just one reliable source. The program as a whole interacts with a single store, eliminating the possibility of several instances leading to data discrepancies. Additionally, it prevents the wasteful resource use that several instances would produce. Access and manipulation of the data are made more efficient.

# End Points

| ID | Endpoint | Verb | Payload | Expected Response | Description |
|---|---|---|---|---|---|
| 1 | /update | POST | N/A | { "map": { "matrix": [...], "score": 0, "isWin": false } } | This endpoint is used to retrieve an updated game map and its current score. |
| 2 | /pacman | POST | N/A | { "pacman": { "life": 3, "x": 10, "y": 20, "direction": "up" } } | This endpoint is used to retrieve information about the Pacman character, including its life, position (x, y), and current direction. |
| 3 | /ghost | POST | N/A | { "ghosts": [ { "color": "red", "x": 28, "y": 19, "direction": "up" }, ... ] } | This endpoint is used to retrieve information about all ghosts in the game. The response is a list of ghost objects, each containing its color, position (x, y), and current direction. |
| 4 | /clear | GET | N/A | N/A | This endpoint does not return a specific JSON response. It clears the game state. |
| 5 | /levels | GET | N/A | {<br>"level": 1<br>} | This endpoint is used to set the game level by providing the desired level as a query parameter. The response indicates the updated game level. |
| 6 | /set | POST | N/A | {<br>"pointsNum": 240,<br>"ghostsNum": 4,<br>"lives": 3<br>} | This endpoint send back the custom settings of points, ghosts and lives number. |

# Ghost Strategies

- Basic: Ghost will see the location of the Pacman and then try to catch the Pacman.
- Fruit Hunter: Prioritizes moving towards the fruit, if available on the map.
- Shy: Moves towards Pac-Man until it gets within a certain distance, then it tries to move away.
- Patroller: This ghost patrols a specific area of the map, changing direction only when hitting a wall.

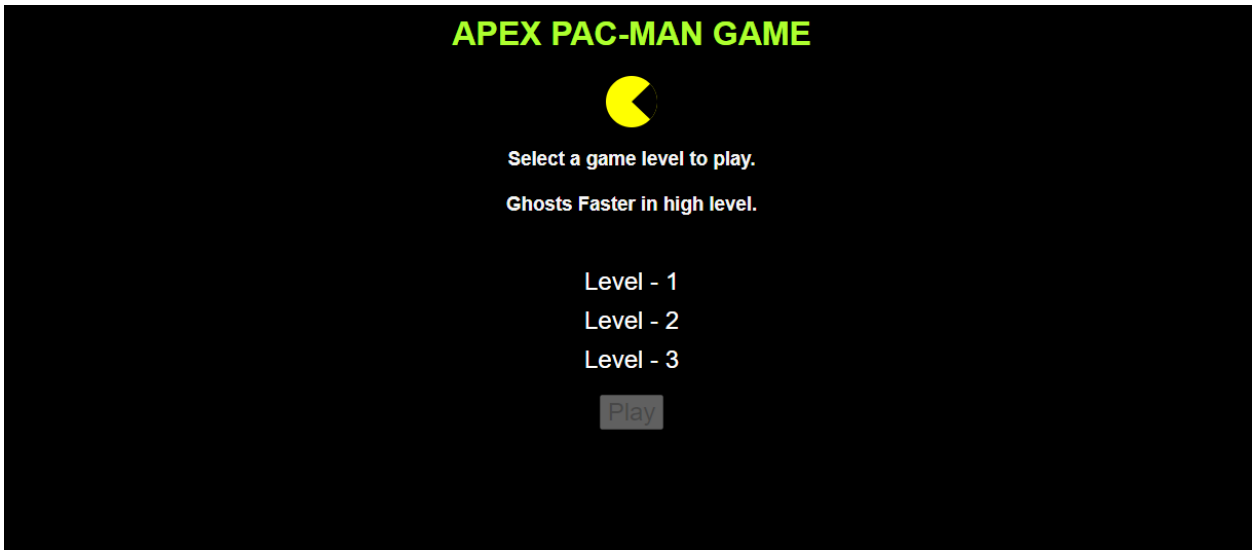
# Map Design

- 0 = Small Points

- 1, 7 = Walls
- 2 = Flash Points
- 3 = Fruit
- -1 = Blanks Can have Fruit

# PacMan Game GUI

Landing page:




Main Page:


Game Over Page: