

Implementation Plan for Pacman Game

Following a logical progression will help ensure that foundational elements are in place before moving on to more complex features. Here's a structured plan:

Phase 1: Setting up the Board

1. Board Layout and Coordinate System:

- Decide on the dimensions of the game board.
- Update the `Position` and `Vector` types to use `Float`.
- Implement any conversion functions if you want to work with both `Int` and `Float`.

2. Wall and Static Elements:

- Define the static layout of the walls using the `Wall` type.
- Render these walls on the board.

3. Food Placement:

- Place the food (`Food` type) items at designated positions on the board.
- Ensure food isn't placed inside walls.

Phase 2: Main Characters and Movement

4. Implement Pacman:

- Create and place Pacman at a starting position.
- Implement movement for Pacman using arrow keys.

5. Ghost Creation and Basic Movement:

- Create ghosts (`Ghost` type) and place them at their starting positions.
- Implement basic random movements for the ghosts to get them moving.

Phase 3: Game Dynamics

6. Pacman and Food Interaction:

- Detect when Pacman is on a `Food` position.
- Increase score and remove the food item from the board upon collision.

7. Pacman and Ghost Interaction:

- Detect collisions between Pacman and any ghost.
- Implement life deduction or game end mechanics based on these interactions.

8. PowerUps:

- Randomly place power-ups (`PowerUp` type) on the board.

- Implement the effect of each power-up (like invulnerability or increased speed) when Pacman consumes them.

Phase 4: Advanced Game Features

9. Advanced Ghost Movement:

- Replace the basic random movement of ghosts with the distance-based pathfinding algorithm.

10. Animations:

- Implement animations for Pacman eating, dying, and consuming power-ups.
- Ensure animations are associated with their respective entities (like attaching an eating animation to Pacman).

11. Randomness & PowerUps:

- Integrate seed-based pseudo-random number generation for power-up placements and any other randomness required in the game.

Phase 5: Game States and External Interactions

12. Pause Mechanism:

- Implement the ability to pause the game using the 'p' key.

13. File IO:

- Set up the ability to save and load high scores.
- Implement customization by loading different icons for Pacman.

14. Game State Management:

- Organize the game state (`GameState` type) to keep track of all dynamic elements of the game.
- Implement game states like "Running", "Paused", "Game Over", etc.

Phase 6: Refinement and Abstraction

15. Implement Typeclasses:

- Define and implement the `Renderable`, `Boundable`, and `Movable` typeclasses.
- Refactor existing code to use these typeclasses, promoting code reusability and abstraction.

16. Testing and Refinement:

- Thoroughly test all game mechanics.
- Refine any features or interactions that feel clunky or unintuitive.

Phase 7: Final Touches

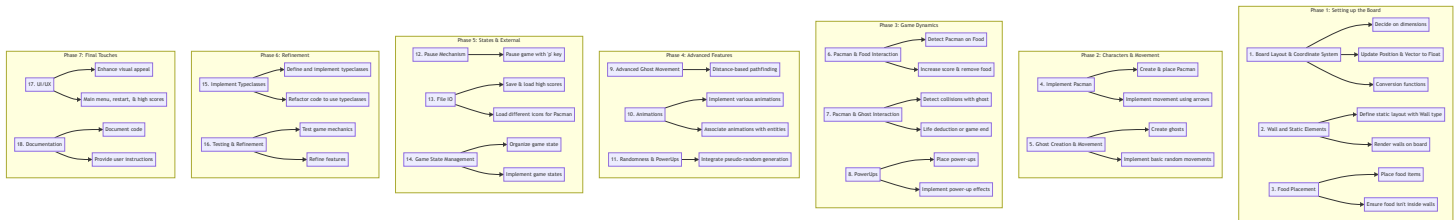
17. UI/UX:

- Enhance the game's visual appeal with better sprites, animations, and possibly background music.
- Add a main menu, options to restart the game, and display high scores.

18. Documentation:

- Document the code comprehensively.
- Provide user instructions and any necessary credits.

Remember, this is a broad plan. Depending on your exact needs and constraints, you may need to adjust it. Some steps may overlap, and as you progress, you might discover additional features you want to implement or find some to be unnecessary.



For the **MVC (Model-View-Controller)** model:

Model

This part will contain the data and the logic of your game.

1. Data Types and Structures:

- All the data types such as `Position` , `Food` , `Wall` , `PowerUp` , `Pacman` , `Ghost` , and `GameState` .
- Logic for updating these data structures, such as moving Pacman or a Ghost, consuming a PowerUp, etc.

2. Game Logic:

- Collision detection between Pacman and other entities (like food or ghosts).
- Effects of power-ups.
- Score computation.
- Game state transitions, e.g., from "Running" to "Paused" or "Game Over".

3. Randomness & PowerUps:

- Logic for the seed-based pseudo-random number generation.

View

This represents the presentation layer, responsible for displaying the game to the player.

1. Rendering:

- Drawing the game board, including walls and static elements.
- Displaying Pacman, ghosts, food, power-ups, and other game entities.
- Showing animations associated with events, like Pacman eating or dying.

2. UI/UX:

- Displaying the game's score, lives left, and other relevant indicators.
- Any additional interfaces like the main menu, options, high scores, etc.

3. Typeclasses for Rendering:

- Implement the `Renderable` typeclass to abstract the rendering of different game entities.

Controller

This will handle user input and manage the flow of the game.

1. User Input:

- Capturing arrow key presses to move Pacman.
- Detecting other key presses, like "p" for pause or other controls you might implement.

2. Game State Management:

- Transitioning between different game states based on user input or game events. For instance, transitioning to "Paused" when "p" is pressed.
- Managing the game loop, where the game repeatedly checks for user input, updates the game state, and then renders the updated state.

3. File IO:

- Handling reading from and writing to files, for features like high scores or customization.

