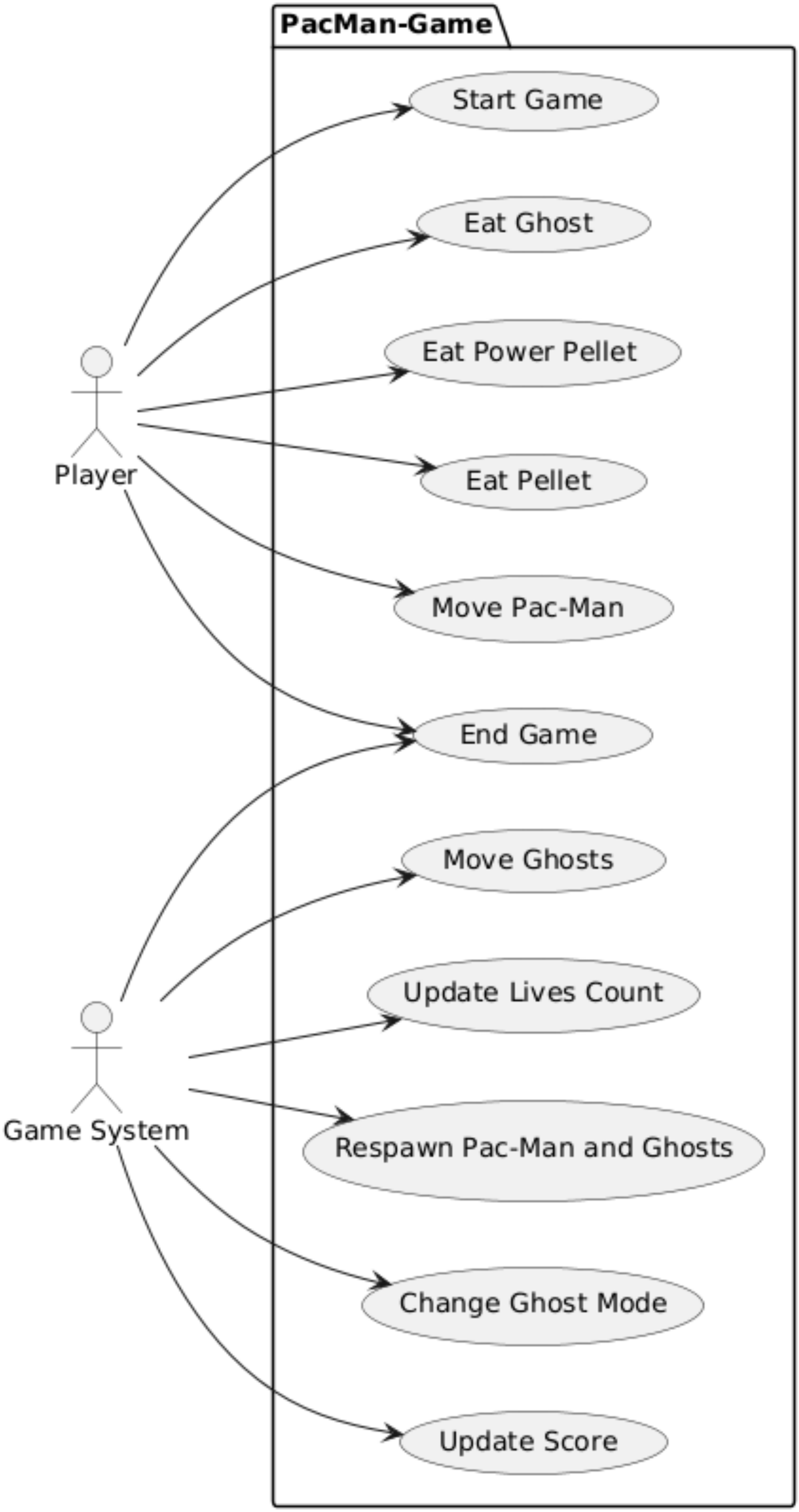
510435765

VIDEO LINK / LINK 2 IF THE OTHER ONE IS BLURRY

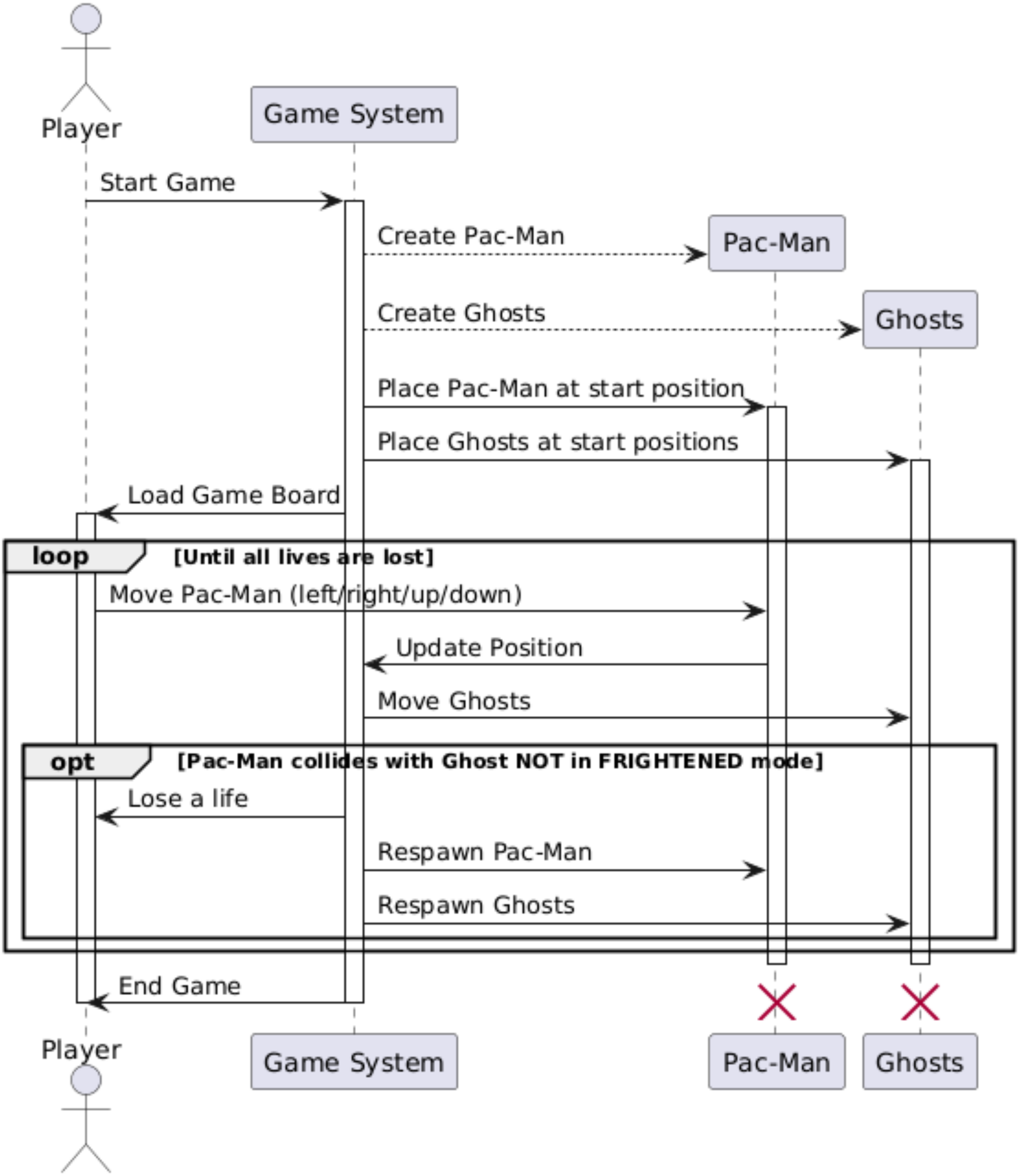
Task 1: Use Case & Use Case Diagram

Use Case Section	**Comment**
Use Case Name	Eat A Ghost
Scope	Pac-Man Game Application
Stakeholders and Interests	**Player**: User wants to achieve a high score by the end of the game, by eating ghosts after consuming a power pellet. **Ghosts**: Ghosts move randomly after Pac-Man consumes a power pellet and will return to the centre of the map after eaten. **Game**: Game systems wants to track Player's/Pac-Man's interactions to update game dynamically, (e.g. score, ghost location and status)
Primary Actor	Player/Pac-Man
Pre-conditions	Pre-conditions include: - Game is currently running and in progress on a level, i.e., the power pellet was not the last dot for Pac-Man to eat on the current level. - Player/Pac-Man has at least one life remaining. - Player/Pac-Man has consumed a power pellet, and the ghosts are now in FRIGHTENED mode to be consumed. - There is at least one ghost in the maze within reach of Player/Pac-Man while FRIGHTENED mode is active.
Success Guarantee (Post-conditions)	Post-conditions include: - Target ghost is eaten by Player/Pac-Man as Pac-Man has come into contact with a ghost in FRIGHTENED mode. - Player's score increases by a minimum score of X. Score of eating Ghost increases as Pac-Man continues to eat more ghosts. - Ghost returns to starting position in SCATTER mode. - Eating animation/sound effect to confirm ghost eaten. - Game continues until all dots are eaten or Pac-Man loses all lives.
Main Success Scenario	1. Player consumes a power-pellet, activating FRIGHTENED mode for all ghosts. 2. Player directs Pac-Man towards a nearby ghost in FRIGHTENED mode using arrow keys for movement. 3. Pac-Man collides with the ghost. 4. Visual and audio confirmation of Pac-Man eating the ghost plays. 5. Player's score increments based on ghost score. 6. Targeted Ghost returns to starting position in SCATTER mode. 7. Game continues with Pac-Man aiming to eat the remaining dots avoiding losing all lives.
Extensions	1. If Ghost avoids Pac-Man/escapes out of reach and Pac-Man cannot eat collide with the Ghost before FRIGHTENED mode expires, the ghost swaps back to SCATTER mode, and Pac-Man can no longer eat this ghost unless Pac-Man can consume another Power Pellet. 2. If Pac-Man collides with multiple ghosts in FRIGHTENED mode simultaneously, system will process each ghost separately, increase the score for each ghost eaten, and return the ghosts to starting position as well as reset to SCATTER mode. 3. Pac-Man collides with a Ghost that was previously eaten and in SCATTER mode (assuming SCATTER mode lasts longer than the duration of FRIGHTENED mode). Pac-Man loses a life, and Pac-Man

	and all ghosts return to starting positions. If Pac-Man has lives remaining, the level restarts otherwise game ends.


PacMan-Game
Start Game
Eat Ghost
Eat Power Pellet
Eat Pellet
Move Pac-Man
End Game
Move Ghosts
Update Lives Count
Respawn Pac-Man and Ghosts
Change Ghost Mode
Update Score
Player
Game System

Task 2: Sequence Diagram



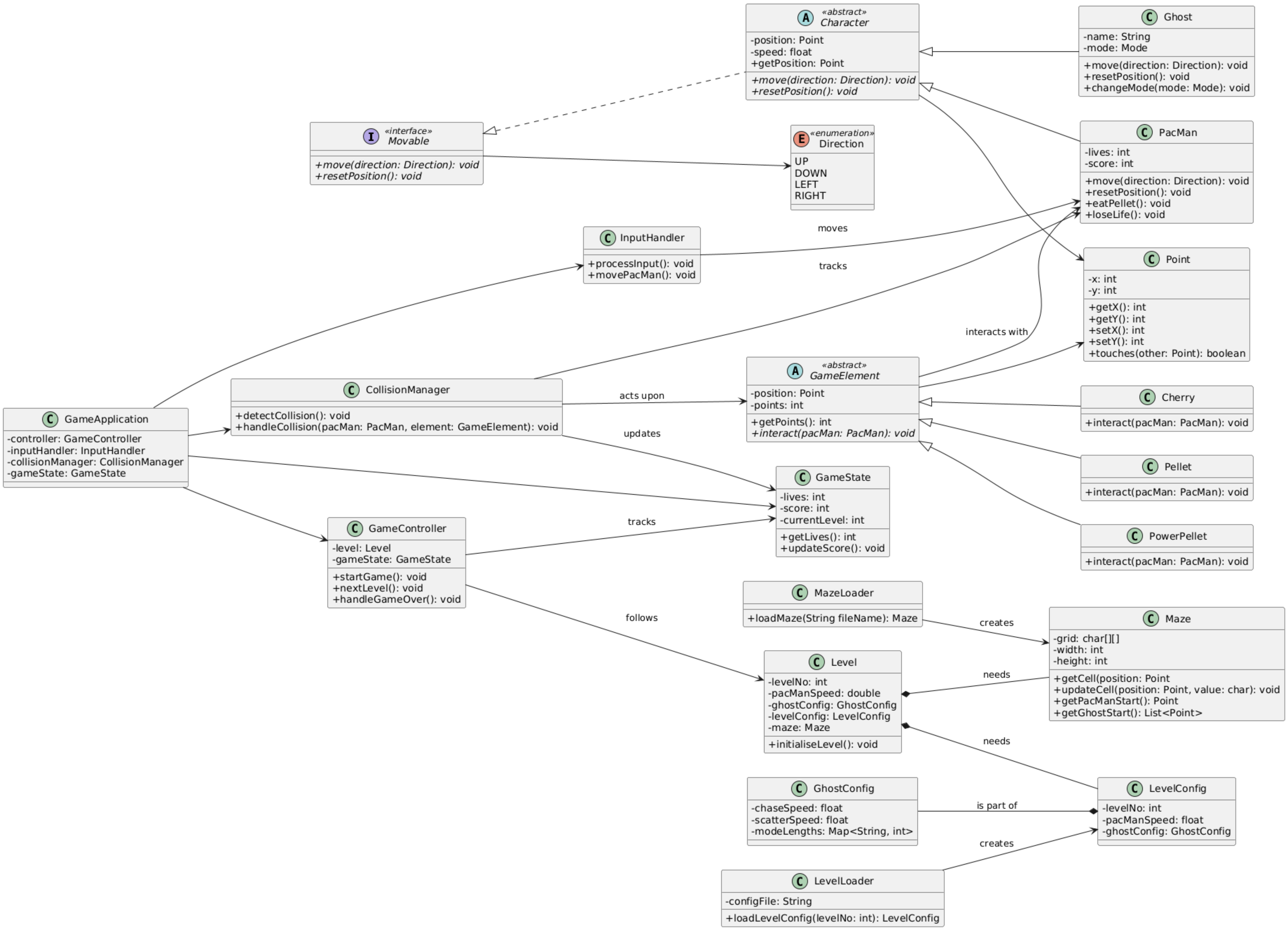
Task 3: Pac-Man Application UML Diagram

This system represents a Pac-Man game, including entities such as PacMan, ghosts, pellets, a maze structure and configuration files which dictate the behaviour of a level.

GameApplication is the central coordinator, not handling game logic directly but instead delegating to other classes, namely GameController, InputHandler, CollisionManager and GameState. GameController. This made sure that we adhered to the Single-Responsibility Principle, as well has making each class an Information Expert. By encapsulating the Core Game Components, we make sure that each class is responsible for specific data and behaviour.

In regards to Game Entities, Character is an abstract class which implements the Movable Interface, representing any entity that can move, namely the two subclasses, PacMan and Ghost. These inherit from Character and share common movement logic. Each character has its own Point, which is an object that tracks position. GameElement is an abstract class representing objects that Pac-Man can interact with, namely the subclasses Pellet, PowerPellet and Cherry, which provide their own implementation of interact(), as they interact with PacMan in different ways. New entities can be added by extending GameElement or Character without modifying existing code, ensuring we adhere to Open/Closed Principle, as well as Liskov Substitution Principle, as subclasses can replace their parent class and the game would function as normal.

Configuration is a separate part of our system. LevelConfig is an object which holds config data specific to each level including Pac-Man's speed as well as Ghost info. GhostConfig holds config data specific to ghosts, and it is a composition of LevelConfig, as each LevelConfig will also have GhostConfig. We encapsulate these in separate classes for further extension if we wanted to change Ghost settings easily. LevelLoader loads level configs from the JSON file. MazeLoader loads a Maze object from the txt file, and Level class represents a ready game level, using LevelConfig and Maze to provide to the GameController. Each class is solely responsible for holding their own config data, or for loading their own config data. Level depends on LevelConfig and Maze instead of the actual loading system, meaning that we can change the config loading process if we have to, satisfying the Dependency Inversion Principle.


«abstract»
Character
-position: Point
-speed: float
+getPosition: Point
+move(direction: Direction): void
+resetPosition(): void
Ghost
-name: String
-mode: Mode
+move(direction: Direction): void
+resetPosition(): void
+changeMode(mode: Mode): void
«interface»
Movable
+move(direction: Direction): void
+resetPosition(): void
«enumeration»
Direction
UP
DOWN
LEFT
RIGHT
PacMan
-lives: int
-score: int
+move(direction: Direction): void
+resetPosition(): void
+eatPellet(): void
+loseLife(): void
InputHandler
+processInput(): void
+movePacMan(): void
moves
tracks
Point
-x: int
-y: int
+getX(): int
+getY(): int
+setX(): int
+setY(): int
+touches(other: Point): boolean
interacts with
«abstract»
GameElement
-position: Point
-points: int
+getPoints(): int
+interact(pacMan: PacMan): void
CollisionManager
+detectCollision(): void
+handleCollision(pacMan: PacMan, element: GameElement): void
acts upon
Cherry
+interact(pacMan: PacMan): void
GameApplication
-controller: GameController
-inputHandler: InputHandler
-collisionManager: CollisionManager
-gameState: GameState
updates
Pellet
+interact(pacMan: PacMan): void
GameState
-lives: int
-score: int
-currentLevel: int
+getLives(): int
+updateScore(): void
PowerPellet
+interact(pacMan: PacMan): void
GameController
-level: Level
-gameState: GameState
+startGame(): void
+nextLevel(): void
+handleGameOver(): void
tracks
MazeLoader
+loadMaze(String fileName): Maze
creates
Maze
-grid: char[][]
-width: int
-height: int
+getCell(position: Point
+updateCell(position: Point, value: char): void
+getPacManStart(): Point
+getGhostStart(): List<Point>
follows
Level
-levelNo: int
-pacManSpeed: double
-ghostConfig: GhostConfig
-levelConfig: LevelConfig
-maze: Maze
+initialiseLevel(): void
needs
needs
GhostConfig
-chaseSpeed: float
-scatterSpeed: float
-modeLengths: Map<String, int>
is part of
LevelConfig
-levelNo: int
-pacManSpeed: float
-ghostConfig: GhostConfig
creates
LevelLoader
-configFile: String
+loadLevelConfig(levelNo: int): LevelConfig

510435765