Group: Chathuni and Urvashi

Pacman game design documentation.

**Step 1: Decomposition**

Purpose:

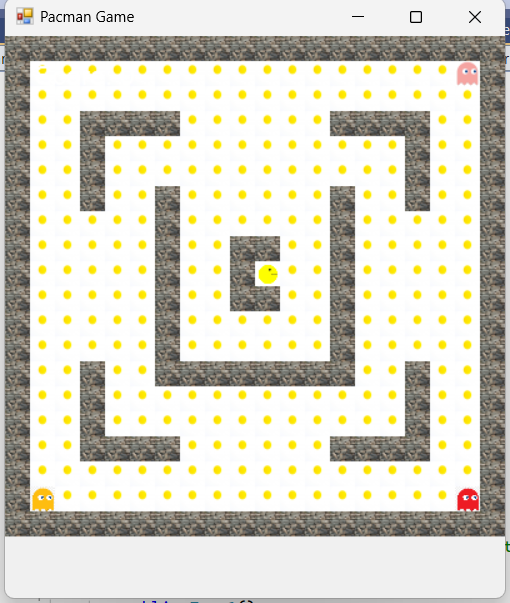
* To create a Pacman game which is smooth, fast and responsive where players can control Pacman to eat kibbles in a maze without running into ghouls.

Description:

* We will start by making walls, Pacman, a maze, and ghouls to set up the game. Set Pacman movement, ghoul movement, and collision detection. Providing the option of pause and resume buttons, tracking the score, and display on the screen.

**Step 2: Form design**

* At the beginning of the game, the design layout will display walls, Pacman, maze, and ghouls.
* It will display both current score and lives.



**Controls and components:**

* Pac-Man Movement (Player Character): Use arrow keys to move Pac-Man up, down, left, and right.
* kibble: It is placed throughout the maze. When Pac-Man eats a maze, it disappears, and the score increases.
* Ghouls: Ghouls move through the maze trying to catch Pac-Man. Each ghost has a different movement pattern
* Score display: Displays the current score, which increases when Pac-Man eats pellets and ghouls.
* Timer: Manages game loops and timing for power-up durations and ghost behavior patterns.
* Game over message: Displays when Pac-Man loses all lives or completes the level

**Step 3: Abstraction**

Identify the classes, I am going to have five classes there are:

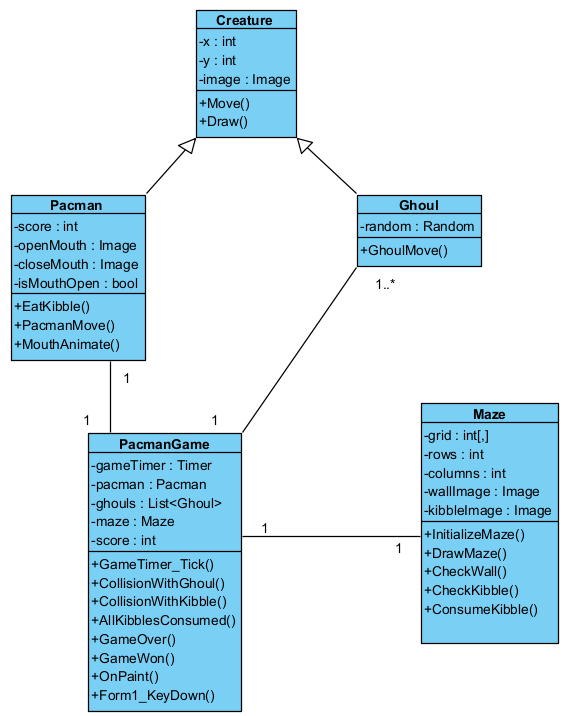
* Pacman (pacman.cs)
* Ghoul (ghoul.cs)
* Maze (maze.cs)
* Creature (creature.cs)
* Pacman game(form1.cs)

The purpose of each class:

* Pacman: To manage the Pacman movement, animation and scoring
* Ghoul: To manage the movement of the ghoul which represents an enemy character that is moving around the maze.
* Maze: To manage the layout and drawing of the maze.
* Creature: This is a base class for ghouls and Pacman.
* Pacman game: To manage overall game logic.

**Step 4: Encapsulation**

This is our UML class diagram



Methods:

Creature

* Move()
* Draw()

Pacman

* EatKibble()
* PacmanMove()
* MouthAnimate()

Ghoul

* GhoulMove()

Maze

* DrawMaze()
* CheckWall()
* CheckKibble()
* ConsumeKibble()

Pacmangame(form1)

* GameTimer\_Tick()
* CollisionWithGhoul()
* CollisionWithKibble()
* AllKibblesConsumed()
* GameOver()
* GameWon()
* Form1\_KeyDown()

Fields:

Creature

* x: int
* y: int
* image: Image

Pacman

* score: int
* openMouth: Image
* closeMouth: Image
* isMouthOpen: bool

Ghoul

* random: Random

Maze

* grid: int[,]
* rows: int
* columns: int
* wallImage: Image
* kibbleImage: Image

Form 1

* gameTimer: Timer
* pacman: Pacman
* ghouls: List<Ghoul>
* maze: Maze
* score: int

**Step 5: Iterative Refinement**

Purpose of each method:

Move() – It will be used to move both Pacman and Ghoul with it’s specific movement for each creature.

Draw() – It will draw the creature at it’s relevant position.

EatKibble() – It will increase the Pacman’s score by 1 when it eats each kibble.

PacmanMove() – It will override the move method for the Pacman.

MouthAnimate() – It will set the Pacman’s mouth for animation.

GhoulMove() – It will override the move method for the Ghoul.

InitializeMaze() – It will set up the maze with walls and kibbles.

DrawMaze() – It will draw the maze on the form.

CheckWall() – It will check if the cell is a wall.

CheckKibble() – It will check if the cell is a kibble.

ConsumeKibble() – It will mark a kibble when it’s consumed by Pacman.

GameTimer\_Tick() – It will handle the timer tick event.

CollisionWithGhoul() – It will check the collision between Pacman and Ghoul.

CollisionWithKibble() – It will check the collision between Pacman and Kibble.

AllKibblesConsumed() – It will check if all the kibbles are consumed in the maze.

GameOver() – It will handle the game over scenario.

GameWon() – It will handle the game won scenario.

OnPaint() – It will draw the game components on the form.

Form1\_KeyDown() – It will move the Pacman using the arrow keys