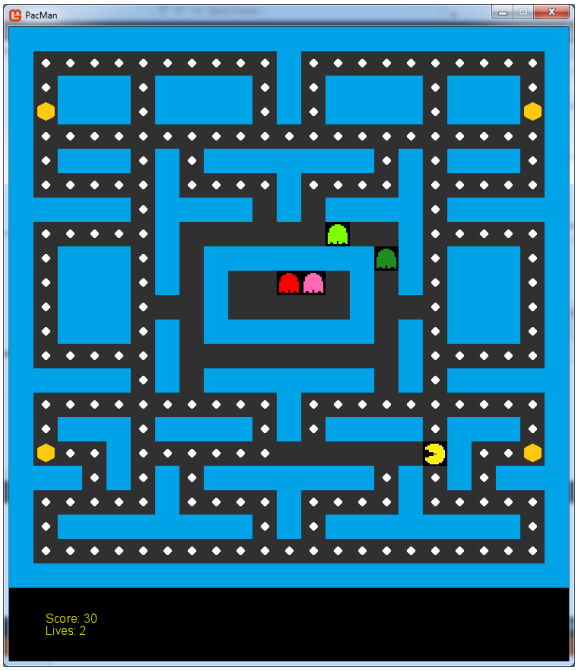
**Sw Devt IV – Advanced .NET  420-411-DW**
**Pacman Project**
**Phase 1 – Business classes**




**Project Phase 1:** Complete implementation of all business classes, and perform complete unit testing

**On deck - Project Phase 2:** Add a Monogame UI which will take care of the game loop, user input and drawing on the screen.

This assignment is heavily inspired by this project from Brown University (an Ivy League university ☺ )

# Introduction (from Brown U)

## Overview of the game

Pacman is played on a 23 x 23 grid organized into a maze. The user controls Pacman's direction of movement with the keyboard. The goal is to eat all of the pellets while avoiding the four ghosts (Blinky, Pinky, Inky, and Clyde) who chase after Pacman.

## Gameplay

### *Pacman*

If Pacman runs into a ghost, he loses a life, and the ghosts are all reset to their original starting locations. If Pacman collides with a dot or energizer, he eats the item, which disappears from the board and updates the score. Eating an energizer sends the ghosts into frightened mode for a short time.

### *Ghosts*

When not in frightened mode, the ghosts move according to a shortest path algorithm which is outlined in the Ghost Behavior section.

In frightened mode, the ghosts' behavior changes (more on this in the Ghost Behavior section), and Pacman is able to eat them. The ghosts all simultaneously change their color to blue when frightened mode begins and revert to their original colors when frightened mode ends. When a ghost is eaten, it respawns in the ghost pen. The ghost pen periodically releases ghosts to the free square just above the pen.

### *Scoring*

Whenever Pacman eats a dot, energizer, or ghost in frightened mode, the score should be updated as follows:

- Dot: +10 points
- Energizer: +100 points

● Frightened Ghost: +200 points

### *Winning and Losing*

The game ends either when either all of the dots and energizers are eaten or when Pacman loses all of his lives. When the game ends, the ghosts and Pacman should stop moving.

### *The maze*

Instead of hard-coding the entire maze, you will read it from a file. The file indicates the content of every tile on the 23 x 23 grid; all objects are instantiated at this time. The GameState class has a static method which is responsible for this initialization. It will end up instantiating the Maze (which represents the grid of Tiles) with all the Pellets and Energizers, the Pacman, the 4 Ghosts in a GhostPack and the Pen. It will also instantiate the ScoreAndLevel class. For simplicity, we have removed the wrapping functionality (i.e., moving from a rightmost tile out of frame and appearing on the left)/

### *Controlling Pacman*

You will use the keyboard to control Pacman's movement – this will be done through Monogame in the second phase. If a directional key is pressed, Pacman begins moving in that direction only if he is free to move in that direction. He *continues* moving in that direction until he either runs into a wall or is told to turn in a new, valid direction. When Pacman runs into a wall, he stops moving.

### *Collision Detection*

Collisions in Pacman are simple: A collision occurs if Pacman's location on the grid is the same as another object's location on the grid. However, the result of a collision differs greatly depending on what object Pacman collides with.

When detecting collisions, there is a common bug that occurs when Pacman and a ghost are moving toward each other. If the timing is right, it is possible for the two to switch cells at the same time. Since they were never in the same cell, the collision isn't detected and Pacman runs right through the ghost!

There is a simple solution to this bug:

1. Move Pacman and check for collisions

2. Move the ghosts and check for collisions

### *Ghost Pen*

When Pacman eats a ghost, it is sent back to the Pen. The ghost pen should release a ghost into the maze at periodic intervals, using a Timer. The ghosts should be released in the same order that they enter the pen, so use a Queue of Ghosts with a List of Timers.

### *Ghost Behavior*

The ghosts have three different modes:

- *Chase:* the ghosts chase a target close to Pacman
- *Penned:* the ghosts don't move - optional
- *Frightened:* the ghosts move randomly

The most important aspect of ghost behavior is that a ghost should *never do a 180-degree turn in any mode*. In other words, a ghost can never choose to go back to the same Tile he was just on. However, when switching to frightened mode, you can force a 180 degree turn (optional).

During Chase mode, each ghost has a different target location, which can be expressed as a Vector2. For example, a target of (0, 2), means the ghost will try to get to the tile that has the same x-coordinate but is two tiles below. The ghost chooses from all the available tiles from his location (except the tile where he just was) – the choice is based on the tile which is closest to the target coordinates. The Vector2 struct has a static method: Vector2.Distance(vectorA, vectorB) that returns a float. Use this to find the closest Tile.

During Frightened mode, simply choose randomly from the available Tiles.

The overall design of the business classes is provided in the last page of this document. Please see your instructor to discuss any design changes that you wish to make. You will notice that the UML class diagram provided is **not** complete: for example, the methods which must be implemented due to an interface, nor all the private instance variables and methods that you may wish to add.

**Notes on the Pacman UML class diagram:**

- the italicized members of the Tile abstract class are abstract. In the derived classes (Path and Wall), you may decide to simply throw NotSupportedExceptions if the functionality is not appropriate. For example, the Wall class will never contain Members, so the property getter and setter should simply throw an exception
- The Wall and Path class do NOT have all the overridden methods indicated!
- The Maze is made up of a 2D array of Tiles. You may use a rectangular or jagged array. It throws the event PacmanWon when it sees that all the Tiles are empty in the CheckMembersLeft() method. It providers a method GetAvailableNeighbours that takes in the current Tile position as well as the current Direction, and returns a list of available Tiles (i.e., not Walls) besides the one that goes backwards. Direction is an Enum of Up, Down, Left and Right.
- The members inside the Path tiles are ICollidables. They can Be Pellets or Energizers. Everytime Pacman move to a Tile, it invokes Collide on the Tile; if the Tile has an ICollidable member, it invokes Collide on the member. An Energizer will scare the GhostPack and fire an event so the score in increased; a Pellet will only fire an event so the score in increased.
- The GameState contains a static method Parse which reads in a file and creates a GameState object. This object manages the remaining objects – it provides get properties to get the Pacman, GhostPack, Maze, Pen and ScoreAndLives objects
- The Ghosts are probably the most complex part of this project. They have two possible states – Chase or Scared (you can consider Penned if you want as well). The state of the Ghost tells you how the ghost should behave. This is a good place to use a State Design Pattern. Instead of cluttering the Ghost class with an Enum to track it's state and a bunch of if statements everywhere, we will use two separate classes and delegate to them. So when the Ghost is frightened, it will use a Scared IGhostState class to figure out which Tile to move to. If it is in Chase mode, it will use a Chase object to decide its next move. The Scared object simply

chooses randomly among the neighbours returned by the Maze's GetAvailableNeighbours method. The Chase object chooses the neighbor whose distance to the target is the smallest.

- The Pen is the second Ghost-related complexity: when Ghosts are in the pen, they wait to be released. As such, the Pen needs to maintain multiple timers (in a List) and as Timer's elapse, the next Ghost in the queue is released to the starting position. You may notice that the Release method's signature is that of an event handler – it is handling the elapse event.
- The GhostPack is useful to treat the Ghosts as a collective: The GhostPack centralizes where they are asked to move, collision checks, reset back to starting point, and all changed to scared mode.
- Finally the Pacman class is told where to move, so it will get a Direction as an argument to its Move method.

**Stay tuned: Upcoming activities that will help you!**

We will add events to the Pong lab!

## Pacman Business classes