

# PACMAN DOCUMENTATION

## BIT PROGRAMMING 2 – ASSIGNMENT 2

Aardhyn Lavender

---

### Problem Description

Pacman is a game where the player controls a yellow mouth directing him to eat pink dots in a simple maze while avoiding ghosts of varied personalities. Each level is completed when Pac-Man eats all the points (and energisers) or loses when entering the same tile as a Ghost type.

The Characters in the game are

- **Pacman** – the protagonist controlled by the player. Directed using arrow keys, Pac-man continues in the direction specified until hitting a wall or changing direction. Pacman can round corners faster by turning early.
- **Ghosts** – Sprites in the game who alternate between ‘chasing’ Pacman and ‘scattering’. while chasing, ghosts use a specialised tactic to track down Pacman, but all ghosts use the same pathfinding algorithm in order to reach him

**Blinky** The red ghost targets Pacman directly.

**Pinky** The Pink ghost that attempts to ambush Pacman getting in front of him

**Clyde** Targets Pacman directly like Blinky, but scatters when he gets too close.

**Inky** The most seemingly sporadic ghost, who teams up with Blinky to corner Pacman.

- In the original Pacman, the game is never won as the final level 256 is corrupted (*due to the byte allocated in memory to store the level overflowing and bit shifting the subsequent game code*) and does not allow all the points to be collected.  
For this rendition, I will end the game after 255 level. There is no win condition  
Given this alteration, the game has no explicit winning condition, rather advocating the pushing of oneself to his or her limits, achieving higher and higher scores.
- The game is lost when Pacman dies with no remaining *lives* – of which there are 3, giving Pacman 4 attempts to complete the game before it restarts (we'll leave the coin insert out of this one!).  
A *life* is lost when Pacman occupies the same tile as any Ghost type.

### Design

Like Breakout, I will make minimum use of WinForms components in favour of ‘pixel perfect’ rendering to the screen as bitmap data to a large image canvas of specified size and scaled resolution. This means omission of the **DataGridView** in favour of simple 2D array math.

The main functionality of the rendering and component management can be transferred from the *assignment one* as it was made abstract in its implementation for this very purpose (This includes the **Game** and **Screen** classes, as well as other utility classes such as **Animation**, **Time**, and **Task**).

The UI for the actual game of Pacman involves merely switching the data stored in tiles around, such as the **lives**, **Score**, **high score**, and **Items**.

To add the suggested ‘extra functionality’ to the game I introduced a **development** mode, which shows the target tiling information for derived ghost types, and prevents Pacman from losing lives. This mode shows off the correct path finding and target tile calculation algorithm that makes each ghost unique.

When the application is run, the user will be able to choose to run the game in standard or development mode.

The Game will use keyboard input to control the menu selector and Pacman (WASD or Arrow Keys), and detects selection from the space bar, or Return/Enter.

## MVP Planning

The Minimum viable product for Pacman (excluding the boilerplate game engine code) will be **Pacman**, and a **world** Class to manage **GameObject** types.

**Game Objects** contain boilerplate information for all objects that are a part of the game. They will know what **Game** their part of, where to find their texture information they have a position, a **screen** to render to, and virtual **Update**, **Draw**, and **Input** functions. Game Objects will eventually be defined as *abstract* when the dependency on their use for debugging purposes ceases.

The **Sprite** class is a derived **GameObject** type that combines a **GameObject** with a **World** type reference to determine the current tile of the object. Sprites also have trajectory and direction information.

The **Pacman** class – a derived **Sprite** requests the directions currently being inputted and sets the Trajectory, accordingly, checking what tile in the **World** his absolute pixel coordinate aligns with and using 2D array math to determine if the **GameObject** of that index has a wall property of *true*.

The **World** class groups **GameObject** together – eventually will be replaced with a **TileObject** derived **GameObject** class – and provides methods to query, add, and remove them safely. Worlds build tile maps based off data stored in *.tmx* files – generated by the application **Tiled**.

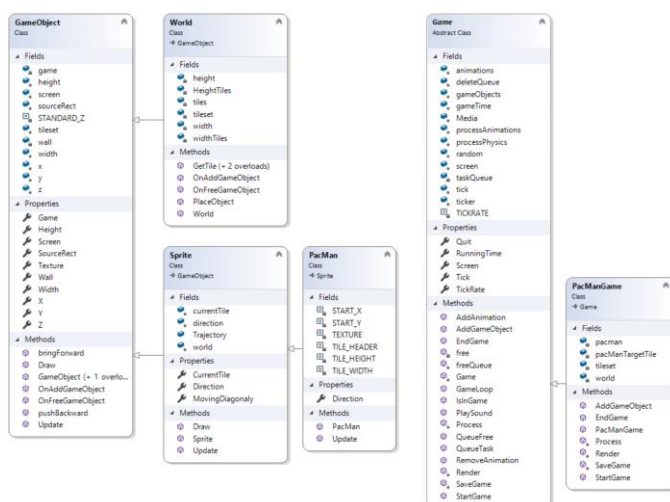
Everything is tied together with the **PacManGame** class which derives the abstract **Game** type. This is the controller class which creates the world, and Pacman, providing entry code for the games start and end

Other classes relating to the object management and rendering are excluded here as their purpose does not strictly relate to *game behaviour*.

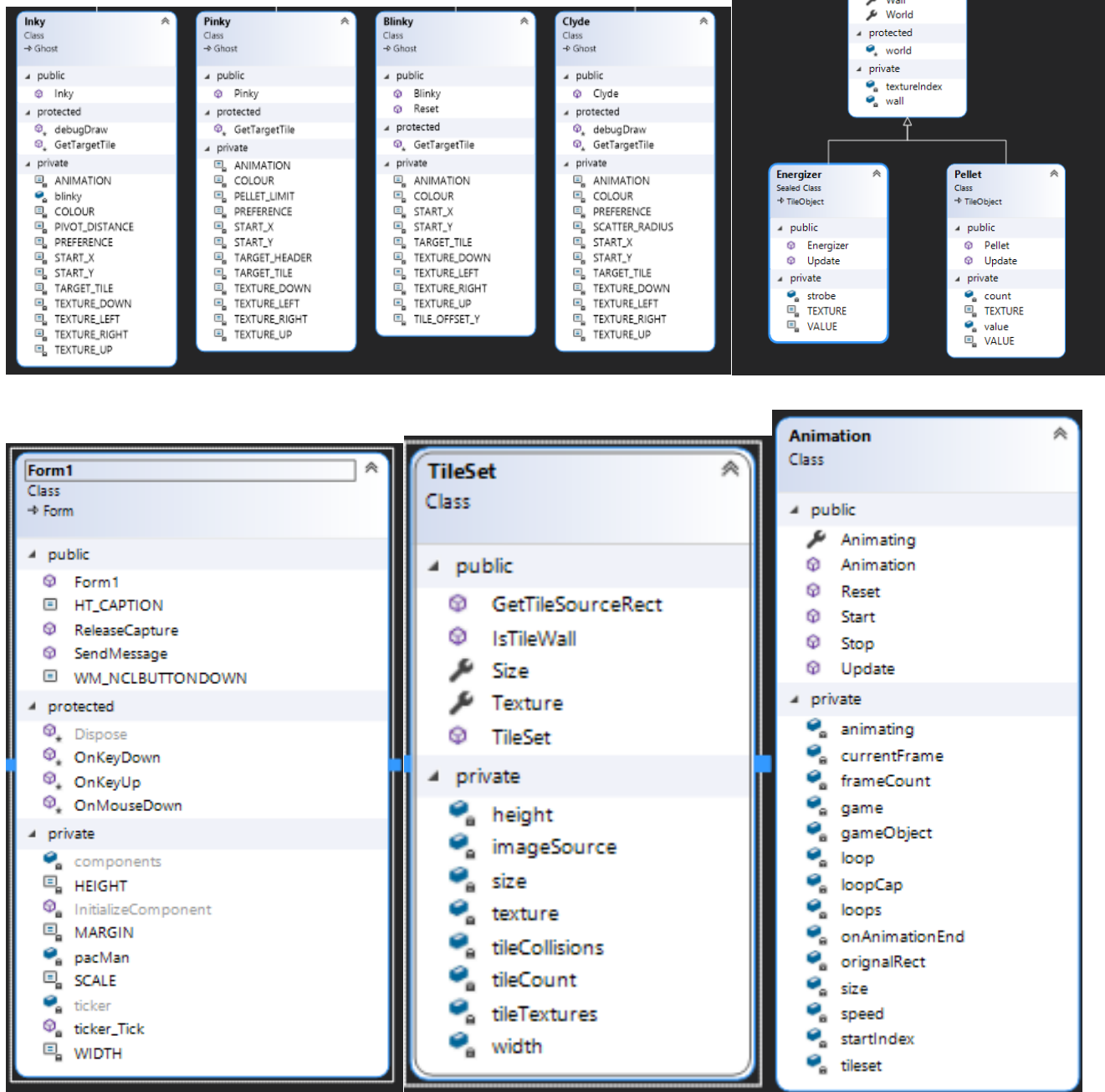
You can view the MVP version of this program with the following git command

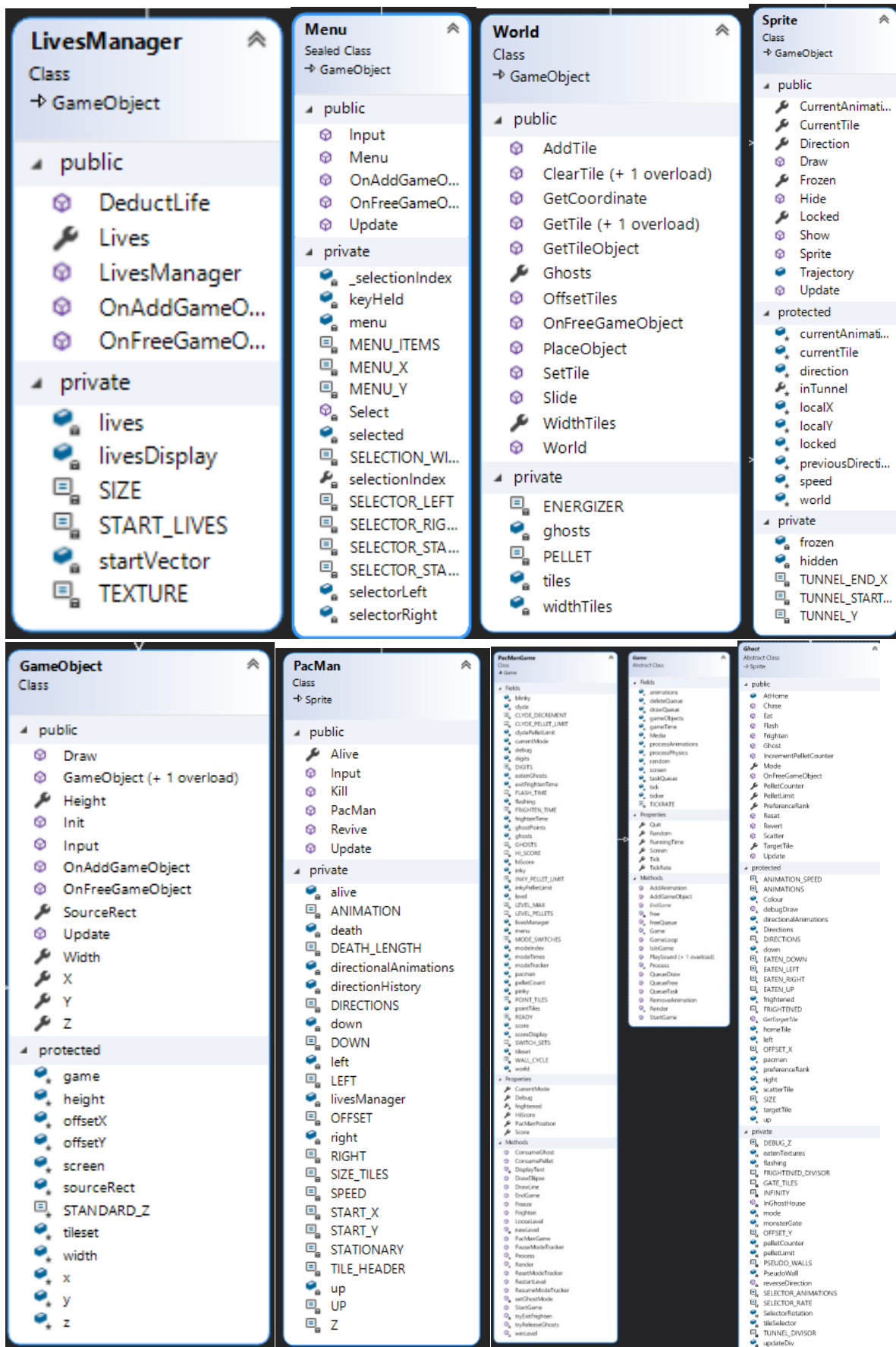
```
git checkout b4711ca88796d8b0a7ea78b898573b2145dc0fe0
```

## Class Diagrams – as of MVP



## Class Diagrams Expanded – Final Version





## Class Inheritance – Final Version

