ECE 462 Individual Programming Assignment

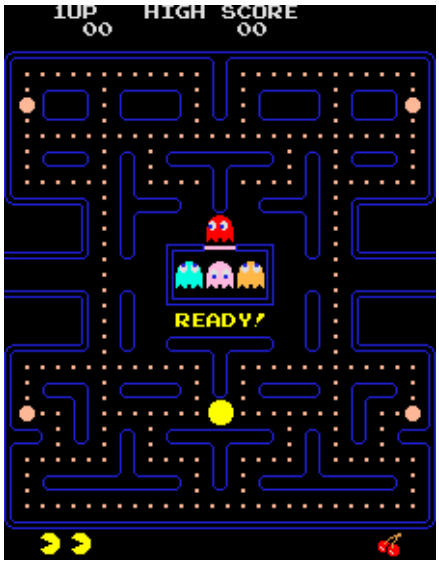
Pac-Man Game

Stable draft: You may start the assignment according to the following specification now. There will not be dramatic changes in this documents, but minor corrections may apply.

In this assignment, you need to write a simplied version of the single-player Pac-Man game. You can use either Java or C++w/Qt.

Pac-Man is an interactive computer game developed in the early 1980s. It was one of the most popular games at that time. It is still being played by many people. The first World Championship was held in 2007 in New York City. You can play the game on-line in several web sites:

- www.pacmangame.net/
- www.learn4good.com/games/pacman.htm
- www.webpacman.com/



Game Components (for this assignment)

The Pac-Man game has a menu and one component on its main window -- the playfield, and everything should be **drawn** into the playfield. The necessary contents of the menu and playfield are listed as follows:

- Menu items: Menu items named "Reset", "Clear High Score","Exit" can respectively restart the game, reset high score to zero and exit the program *at any time*.
- Title screen: A title screen is displayed before the game is started. You can design the title screen freely, but you must include instructions such as "Press any key to start". You may also want to put the how-to-play information within the title screen.
- Characters: There should be 3 moving characters in the game, the Pac-man and two ghosts (in arbitrary colors, but should be at least visible).
- Maze and pac-dots: The layout of the maze is open to your own design, but it should contain **at least four horizontal and four vertical passages**. All passages must be one-character wide, which basically means that the characters can only move *along* the passage. The maze are filled with small dots known as pac-dots except for the initial locations of the characters. For simplicity, power pellets (which enables Pac-man to eat ghosts) are **not** required in this assignment

the characters. For simplicity, power pellets (which enables Pac-man to eat ghosts) are **not** required in this assignment.

- Messages: On the top of the playfield, there are two labels indicating the current score and the high score. The high score should be kept in a persistent storage (i.e. a file). On the bottom of the playfield, the number of rest lives is displayed as a corresponding number of Pacmans on the left. Fruit is not necessary in this assignment.

Game Rules

Most game rules of the original game can be found in Wikipedia. In this assignment, some of the original rules are *eliminated or altered*. A list of required rules are as below (bold texts indicate the modified rules for this assignment):

- Basic gameplay: The player controls Pac-man through a maze, eating pac-dots. **Two ghosts** roam the maze, trying to catch Pac-Man (however, no intelligent strategy is needed). If a ghost touches Pac-Man, a life is lost. The initial number of lives is three.
- Character behaviors: The ghosts *never* turn back halfway, nor do they turn back at crossroads. Pac-Man moves at a constant speed. Player can use keyboard to control the Pac-Man's moving direction, but not speed. If no key is pressed, the Pac-Man continues going forward until it meets a wall and then stops. If two ghosts meet, they keep on moving forward and cross each other.
- Scores: Each pac-dot is worth 10 pts.
- Completing a level: When all pac-dots are eaten, the level is completed and **a congratulation screen is displayed within the playfield**. If the player's score is higher than the stored high score, the new high score is stored. No signature is needed for the high score. After the player hits any key, the game is reset to the title screen.
- Game over: When all lives have been lost, the game ends **and a game over screen is shown in the playfield**. If the player's score is higher than the stored high score, the new high score is stored. After the player hits any key, the game is reset to the title screen.

You may add any functionality as you want to improve the specification, but you MUST first implement all the rules listed above as a minimal set of requirements.

Grading Criteria

Program Functionality (4pt)

1. **(0.2pt)** Main window: A main window appears after the program is launched. The window contains menu items and a playfield. **No other components on the window are allowed** (any function related to these extra components will not be graded).
2. **(0.4pt)** Menu items: "Reset" menu item restarts the game. "Clear High Score" erases the high score record. "Exit" cause the program to quit. They are functional at any time.
3. **(0.2pt)** Title screen: When the main window is shown, a title screen is automatically displayed within the playfield. The title screen should contain the name (preferably the logo) of the game, and an instruction telling player how to start and operate the game.
4. **(0.4pt)** Basic rendering: A maze whose *layout* contains at least four horizontal and four vertical passages is rendered inside the playfield when the game is started with a **black** background. Three characters, Pac-Man and two ghosts in arbitrary colors, are drawn inside the maze at arbitrary initial positions. Other required maze elements include pac-dots and walls.
5. **(0.2pt)** Text: The current player's score and highest scores are displayed at the top of the playfield. In the lower-left corner a number of Pac-Mans is drawn, displaying the number of lives left.
6. **(0.4pt)** Control: Pac-Man can be controlled by keyboard. It can move in the direction as the player instructs, but the speed is invariant. Pac-Man keeps on moving without changing its direction if there is no operation and stops if it meets a wall.
7. **(0.6pt)** Basic movement: All characters move along the lanes and cannot go into unreachable areas which are enclosed by the walls. When a Pac-Man passes a pac-dot (including power pellet), that dot disappears.
8. **(0.4pt)** Animations: When the characters are moving, they are rendered as animations (changing pictures) instead of still images (e.g.  and ). If the Pac-Man stops moving, its image becomes still. If Pac-Man bumps into a ghost, an animation is played (e.g. ). The animations and figures are *by no means* graded by their aesthetical feeling.
9. **(0.2pt)** Behavior of enemies: The ghosts keep moving in the same direction until they reach a cross where they can make a random choice.
10. **(0.2pt)** Game status: Game score, high score and lives left are updated and rendered correctly. The high score is kept in a persistent way so that it won't be lost as the program exits. Game status is initialized whenever a new game is started.
11. **(0.2pt)** Gameover: When all pac-dots are eaten, a congratulation screen is displayed. If all lives are lost, a gameover screen is displayed. In either case, the player can dismiss the screen by hitting a key and the game goes back to the title screen.
12. **(0.6pt) Demo mode** (a.k.a attract mode): When a title screen is idle for 5 seconds (no keyboard input), the game enters demo mode, in which the Pac-Man is controlled by computer. No AI is needed for the computer player. Whenever a key is pressed in demo mode, the game returns to the title screen.

is pressed in demo mode, the game returns to the title screen.

Program Readability (1pt)

Code Standards (0.4pt)

1. **(0.2pt)** The code is formatted uniformly (use the function in your IDE). Classes, objects and members are named nominatively. This is a C++ coding standard for your reference.
2. **(0.2pt)** There should be explanatory comments to help readers understand your code. Give a brief description about the class at the beginning of each class file.

Documentation (0.6pt)

1. **(0.4pt)** Explain what the classes and objects are involved in this program and their functions and **give necessary UML diagrams** (they are graded). You should particularly focus on inheritance and polymorphism (if there is any), explaining how and why you use them.
2. **(0.1pt)** Explain *with sufficient details* the data structure you used for storing pac-dots, maze, etc.
3. **(0.1pt)** Explain your implementation for animation, double buffering, and other rendering issues. Specify the classes and objects you used in relation to these issues.

In addition to the above checkpoints, please list **everything** you have implemented in your documentation. *If something is implemented in the source code but not mentioned in your documentation, it is not graded and you will not receive any point.*

Your documentations can be in one of two formats (1) PDF or (2) README. ***A Word file is not accepted*** because there are too many versions of Word programs and the formatting can be different. README is an ASCII file and the name must be README. This is a widely accepted convention by millions of people. The file name is *not* readme, readme.txt, Readme, or anything else. It must be README. If you writes README file and what to include UML diagrams, you can submit the UML diagrams as separated **PDF files**. Any other formats, such as pictures, Visio files, NetBeans diagrams etc., are not accepted. You can use "Print to File" or PDFCreator to export your diagrams as PDF files.

If you submit a PDF documentation, please embed the UML diagrams into the documentation.

Avoid plagiarism: Whenever you cite text, pictures, ideas, etc., be sure you include the source in your reference. This includes information from the Internet.

Bonus (2pt)

1. **(0.5pt)** Use of sound: If you use any sound in this assignment, you may get bonus points. **Please note that you have accomplished this bonus task in your documentation.**
2. **(1.5pt)** Mimicking the original game: You can obtain bonus points if your Pac-man game is improved to have power pellets, ghost box, teleport tunnel, etc., which is not specified in this assignment but present in the original Pac-man game.

Supplementary Material

Here is a sprite set from the original Pac-Man arcade game by Namco. This is only for your reference and you do not *have* to use it. Since this is a copyrighted material, please do **not** use it for purposes other than this assignment. You need to divide, resize and make transparent the following image by yourself. You can use external image editors, such as GIMP, or keep it as a whole image and "divide" it within your code.