Course project Documentation

CS101 PROJECT

PACMAN GAME

Soham Gaunekar – 140110001
Soumya Nimodiya – 140110051
Nisarg Gagrani – 140110021
Aditya Kumawat – 140110070

Table Of Contents

1. Introduction.....

2. Problem Statement.....

3. Requirements.....

4. Implementation.....

5. Testing Strategy and Data.....

6. Discussion of System.....

7. Future Work.....

8. Conclusion.....

9. References.....

INTRODUCTION

Pacman is one of the most popular arcade games of all time. The player controls Pac – man through a maze, eating pac – dots (also called pellets).

Four ghosts (Blinky, Pinky, Inky and Clyde) roam the maze, trying to catch Pac-Man. If a ghost touches Pac-Man, the game ends and the score obtained is displayed on the screen.

Near the corners of the maze are four larger, flashing dots known as power pellets that provide Pac-Man with the temporary ability to eat the ghosts. The ghosts turn deep blue, reverse direction and usually move more slowly.

This project being largely graphic intensive, we decided to use the Allegro 5 game engine for its development.

The game of PACMAN is mainly meant for recreational purposes, for gamers of all ages.

PROBLEM STATEMENT

The aim of our project was to recreate the classic game of PACMAN, using Allegro 5 and C++. Pacman features a hero controlled using the arrow keys (W/A/S/D may also be used) to manoeuvre it through a maze haunted by four ghosts.

The first goal of this project was to understand the basic functionality of Allegro 5 and it's implementation.

We then had to get the animated hero responding to user input.

The next part of the project was to get the sprites responding to the maze boundaries and stay confined within the maze walls.

The last and final phase involved the incorporation of the ghosts, and animating their movement, using computer generated moves independent of user input.

Requirements

1. Minimum 512 MB RAM
2. 2GB free flash memory
3. Keyboard
4. Mouse
5. Code Blocks IDE linked with Allegro 5

Implementation

The game runs in a while loop with a timer running at 60 frames per second. Every time the timer clicks, the maze, coins, and sprites are redrawn. User input is taken from the keyboard and appropriate functions are called as described below. The entire maze has been represented as a two-dimensional matrix, with each element representing a corresponding 20x20 pixel tile on the game display. The position of each sprite on this matrix is tracked and checked for sprite movement. Sound effects and multiple ghost modes have also been incorporated for a complete user experience.

Header files:

includes.h: contains all the required header files and allegro libraries, along with the global variables. Each of these has been explained in the code comments.

sprite.h: contains the sprite class described below.

startGame.h: contains the startG(ALLEGRO_DISPLAY *display) function used to display the home page of the PACMAN game, giving the user options to start the game, view instructions to play the game, and quit game. This also contains the instructions.h header file included.

instructions.h: contains the instructions(ALLEGRO_DISPLAY *display) function. This is used to display the instructions page and gives the user the option to go back to the main screen.

endGame.h: contains the endG(ALLEGRO_DISPLAY *display) function. This is called when the game gets over and displays the score, credits, and option to quit the game.
checkStatus.h: contains the checkStatus(const sprite &pacman, sprite &ghost) function which checks whether pacman and any of the ghosts occupy the same tile on the map and resets the ghost coordinated accordingly.

redraw.h: used to redraw the maze, coins, and sprites when required. This header file contains the redraw_coins(), redraw_hero(), and redraw_ghosts() functions whose purpose has been described in the code comments.

Class sprite:

This class contains all the relevant attributes and functions to control each character (also known as sprite in the game). Objects of this class are created to represent each sprite in the game. All the attributes have been explained in the comments in the source code.

Move(): This function takes in the values inputted by the user and moves each sprite accordingly, based on direction. This also controls ghost movement by calling the obstructed(int d), turn(), and junction() functions.

obstructed(int d): This function takes in the direction of each sprite and checks to see whether it's movement is obstructed by a maze boundary walls. Returns true if movement is obstructed, else returns false.

junction(): This function is called to check whether the ghost has reached a junction in the maze or not. Returns true if the ghost has reached a junction and then calls turn(). Else returns false.

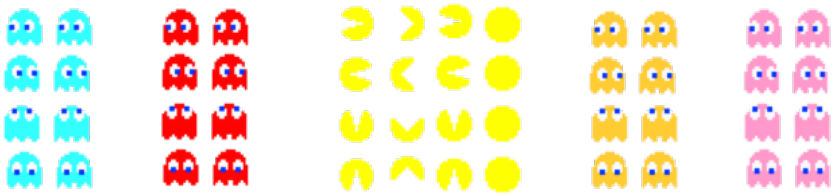
turn(): This function is called every time a ghost has to turn in the maze. This function returns an unobstructed direction for the ghost to move in.

Challenges

Since none of the members had any experience using graphics libraries, the first challenging part of this project was the understanding the functionality of allegro 5 and its libraries.

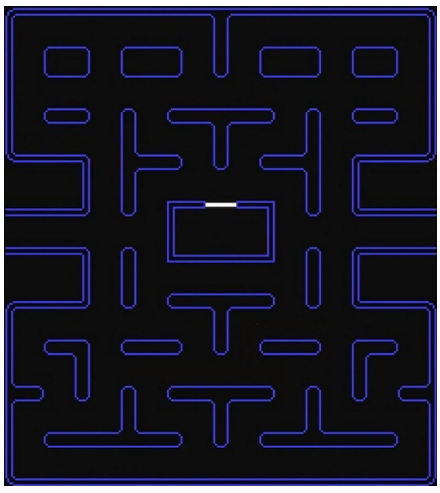
Sprite animation: This was another challenge in setting up smooth, seamless animation of each individual sprite in the game. To implement this feature, we used spritesheet animation and updated a flag variable running in conjunction with the main allegro timer to update the bitmap coordinates of each spritesheet.

The following were the spritesheets used for animating individual sprites:



The biggest challenge however, was getting the sprites to respond to the maze boundaries and stay confined within the walls.

This is the map that the sprites move on:



Due to the large number of maze walls, a unique boundary condition could not be used for each. We thus decided to represent the maze in a two dimensional matrix, each element representing a corresponding 20x20 pixel tile on the maze. 0s represent boundaries while non-zero values represent the legal path for the sprites to move over. Using the direction inputted by the user, the software predicts the tile that the sprite is about to enter. If it has a zero value, the sprite is not allowed to move ahead.

KNOWN BUGS

The only known bug that we reported is that a small part of the sprite bitmap occasionally enters the boundary wall while turning at junctions in the middle of the maze.

FUTURE WORK

An A* search or Depth First Search (DFS) may be incorporated in the code for more comprehensive ghost movement by making use of heuristics.

Multiple levels and random maze generators may also be incorporated for a complete user experience.

The pacman game may also be added as an easter egg in real world applications as recently done by Google in their Google Maps application.

REFERENCES

1. Introduction to programming through C++ by Abhiram G. Ranade
2. Coding made easy video tutorials on YouTube.