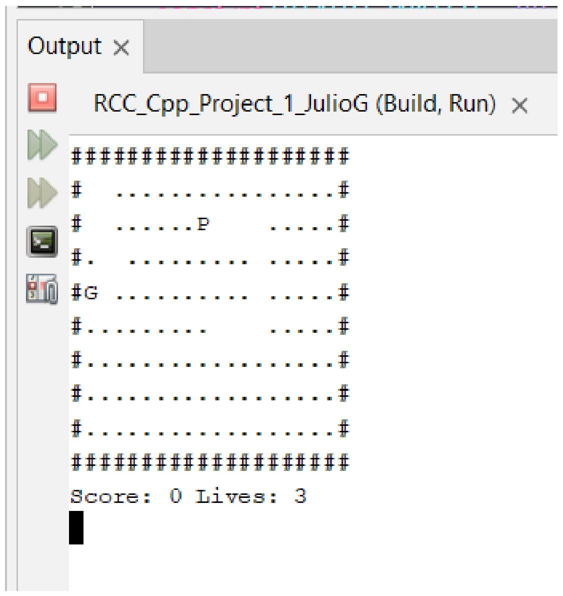
1/30/2024

Introduction

Welcome to Pac-Man version 1.0 game, developed in C++. This program is a text-based implementation of the classic Pac-Man game. It involves moving a player character (Pac-Man) in a 2D grid to collect pellets while avoiding ghosts. The game keeps track of the player's score and lives.

Here's a screenshot of the game.



```
####################
#  ................#
#  ......P    .....#
#.  ......... .....#
#G .......... .....#
#.........    .....#
#..................#
#..................#
#..................#
####################
Score: 0 Lives: 3
```

Summary

The Pac-Man game is a console-based application where the player controls Pac-Man in a 20x10 grid representing the game maze. I tried bigger sizes but anything bigger tends to glitch the screen. The maze contains pellets represented by dots ('.') that the player needs to collect to increase their score. The player's objective is to collect all pellets while avoiding a ghost, represented by 'G', moving randomly in the maze. The player has a limited number of lives, and the game ends if the player loses all lives or collects all pellets.

Description

Key Features:

Game Board: A 20x10 grid where '#' represents the walls, spaces are empty areas, and '.' are pellets.

Player Movement: The player can move Pac-Man using the 'w', 'a', 's', 'd' keys for up, left, down, and right movements, respectively.

Score and Lives: The game keeps track of the player's score, which increases with each pellet collected, and the number of lives.

Ghost Movement: A ghost moves randomly in the maze, creating a challenge for the player.

1/30/2024

Technical Overview:

InitializeGame: Sets up the game board, player, and ghost positions, and initializes score and lives.

DisplayBoard: Renders the current state of the game board in the console.

GetInput: Handles player input for movement.

UpdateGame: Updates the game state, including player and ghost positions, and checks for pellet collection.

MoveGhosts: Handles the logic for the ghost's movement.

CheckWinLose: Determines the game's end condition, either by winning (collecting all pellets) or losing (running out of lives).

EndGame: Displays the game over message and concludes the gameplay.

I really love Pac-Man since I was a kid so making this game was kind of cool. Hopefully it showcases some functions, loops, and decisions in C++. In the game, you get to move Pac-Man around, collect dots, and avoid a ghost. As you play, you'll see how loops (which repeat actions), conditionals (which make decisions), and data structures (which organize information) are used in building the game.

Bugs:

- Ghosts eats some of the dots whenever I move Pac-Man. This shouldn’t happen.
- The score is not changing whenever Pac-Man is eating the dots.
- When you run out of lives you start going into a negative count of lives. Unlimited lives.
- Once the previous bug is fixed I’ll need to create a nice game over message.

Flowchart for CLI-Based Pac-Man Game V1.0

1. **Initialization**

- Set up the game board
- Initialize player (Pac-Man) position
- Place ghosts on the board
- Initialize score and lives

2. **Game Loop**

- Display the game board
- Ask the player for input (movement commands)
- Update Pac-Man's position based on input
- Check for collision with ghosts
  - If collision: reduce a life, reset positions
- Update ghost positions (simple AI for movement)
- Check for Pac-Man eating pellets
  - If pellets eaten: increase score
- Check if all pellets are eaten
  - If yes: player wins, end game
- Check if lives are zero
  - If yes: player loses, end game

3. **End of Game**

- Display final score
- Show win/lose message
- Option to restart or exit

Basic Structure in C++

1. **Global Variables**

- Game board (2D array)
- Player position, score, lives
- Ghost positions

2. **Functions**

- InitializeGame(): Set up the game
- DisplayBoard(): Print the game board to the console
- GetInput(): Get player's movement input

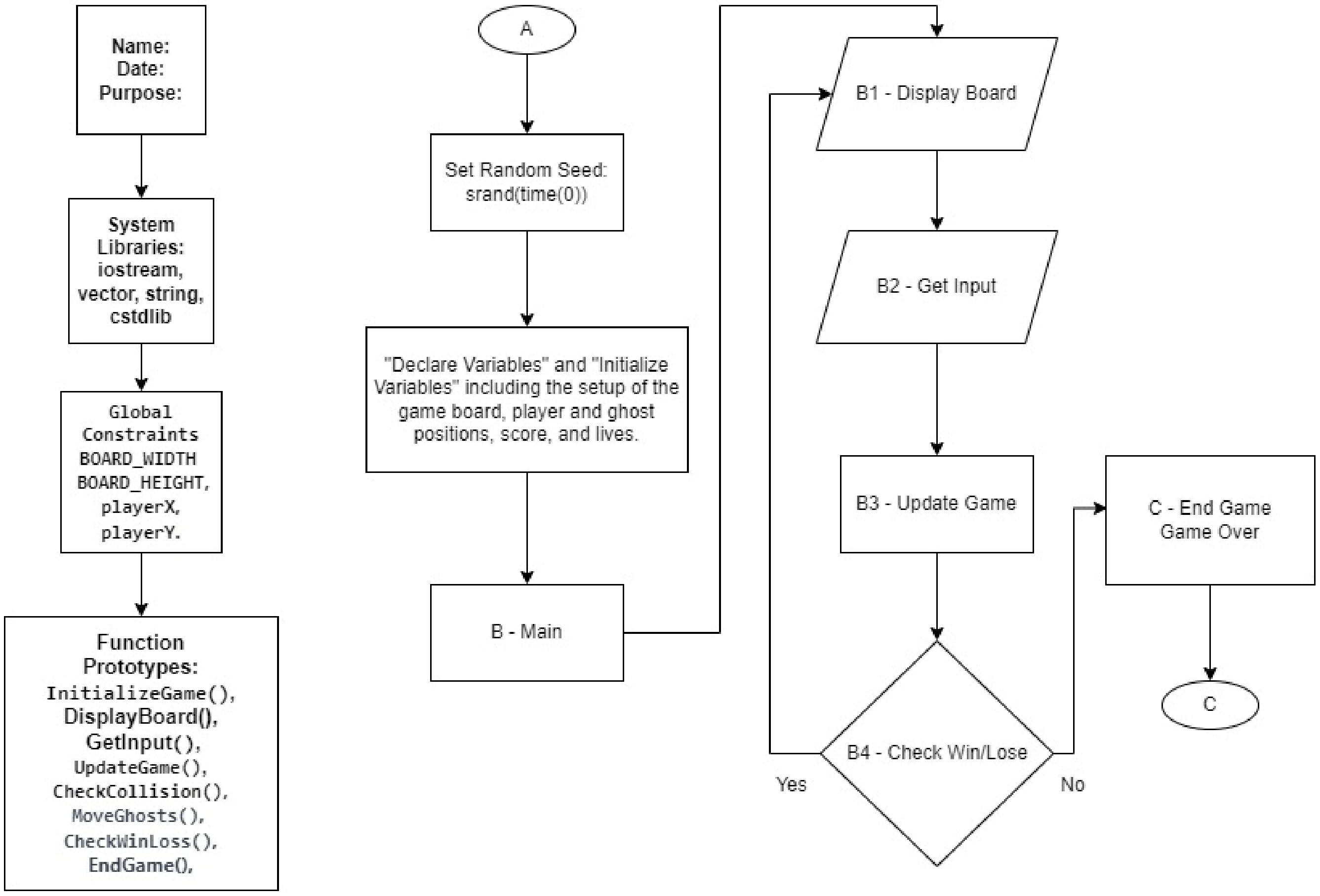
- UpdateGame(): Update all game elements based on input
- CheckCollision(): Check for collisions with ghosts
- MoveGhosts(): Simple AI for ghost movement
- CheckWinLose(): Check if the player has won or lost
- EndGame(): Display end game message and score

3. Main Loop

- While game is not over:
  - Display board
  - Get input
  - Update game state
  - Check win/lose conditions

Type	Variable Name	Description	Location
int	BOARD_WIDTH	The width of the game board	Global
int	BOARD_HEIGHT	The height of the game board	Global
vector<string>	gameBoard	A vector representing the game board where the game takes place	Global
int	playerX	The X-coordinate of the player's position	Global
int	playerY	The Y-coordinate of the player's position	Global
int	score	The current score of the player	Global
int	lives	The number of lives the player has	Global
int	ghostX	The X-coordinate of the ghost's position	Global
int	ghostY	The Y-coordinate of the ghost's position	Global
void	InitializeGame()	Function to initialize the game settings	Function Definition
void	DisplayBoard()	Function to display the game board on the screen	Function Definition
void	GetInput()	Function to get the player's input for movement	Function Definition
void	UpdateGame()	Function to update the game state after each move	Function Definition
void	CheckCollision()	Function to check for collisions between the player and ghosts	Function Definition
void	MoveGhosts()	Function to move the ghosts in the game	Function Definition
bool	CheckWinLose()	Function to check whether the player has won or lost the game	Function Definition
void	EndGame()	Function to execute the end of the game procedures	Function Definition

1/30/2024


Name:
Date:
Purpose:
System Libraries: iostream, vector, string, cstdlib
Global Constraints BOARD_WIDTH BOARD_HEIGHT, playerX, playerY.
Function Prototypes: InitializeGame(), DisplayBoard(), GetInput(), UpdateGame(), CheckCollision(), MoveGhosts(), CheckWinLoss(), EndGame(),
A
Set Random Seed: srand(time(0))
"Declare Variables" and "Initialize Variables" including the setup of the game board, player and ghost positions, score, and lives.
B - Main
B1 - Display Board
B2 - Get Input
B3 - Update Game
B4 - Check Win/Lose
Yes
No
C - End Game Game Over
C