CO324 PROJECT

PACMAN MULTIPLAYER

REPORT

GAMLATH P.G.R.K.D. (E/13/110)

KANEWALA U.C.H. (E/13/175)

GROUP 10

SEMESTER 5

12/05/2017

PacMan Multiplayer Game

Design

This game was designed using the Model View Controller (MVC) architecture. Four classes were created in the backend in order to achieve this design. Those classes are as follows:

- PacManBoard
- PacManPlayer
- PacManGame
- PacManController

A single JavaScript script namely script.js was used to update the board, scores and players for the user's view on the index.html page. How the design fits into the MVC architecture is as follows:

- **Model**
  - PacManBoard
  - PacManPlayer
  - PacManGame

- **View**
  - script.js
  - index.html

- **Controller**
  - PacManController

Implementation

Implementation details of the above design components are explained below.

PacManBoard Class

This class is used to represent the game board data. It contains purely the board parameters. Board width, height and a data structure containing the food distribution on the board are the main parameters in this class. As the data structure to store the food placement on the board, we have used a HashMap in order for easier access to the coordinates on the board and as no duplicate keys are present. A unique integer key has been generated using the specific x and y coordinates of the board and the value is stored as a String matching to the colours of the food as specified in the required JASON object String in event data.

PacManPlayer Class

This class contains all the parameters and functions relevant to a single player. This includes player's name, score, position and other relevant functions. This also handles the movement of the player ensuring the player does not go out of bounds. The player will spawn on the other side of the wall when the player hits the wall.

PacManGame Class

This class contains the game logic for the PacMan multiplayer game. This class combines the separate game board and players together in order to get the needed functionality. This game logic handles the player and game board, ensuring that invalid states such as an out of grid player are not possible. When players collide they respawn into separate random places. This is a standalone game logic which can be used in any PacMan game. This class represents a single state of the game until an event occurs. The getBoardState() function in this class generates the JSON response format required by the frontend using the data at a specific state of the game.

PacManController Class

This class is the servlet class that initiates the whole PacMan game and handles the requests coming from the multiplayer. This class creates new sessions for the first four players that join the game. More than four players are not permitted to enter the game. If another player tries to join the game after all four players have joined, that player will be able to view the current ongoing game without any disturbance to the game. When a player presses a key, the HTTP POST request sent by the client will be handled by the doPost() function in this class. The game is updated by HTTP Server-Send Events. All the required methods are synchronised to handle multiple user requests without any issue.

Script.js

This is the client side JavaScript which handles the operations of updating the game board, sending user keystrokes to the server and displaying the updated game board to the user. It is able to read JSON string which is sent by the PacManGame and display the current board. Also, it displays the current score of all the players.

Gameplay

Four players are required to join in order to begin the game as provided in the specification. In order for a player to join the game, it is required to press any key after loading the game board. A message is displayed to the users asking to wait until joining the rest of the required number of players. The game is finished when all the dots in the board is finished. After all, dots are finished the player with the highest score wins.

How the game can be more efficient

In this implementation, the whole canvas of the game board is redrawn every time a change in the game board happens. That is every time a key stroke is registered at the client's side. This way redundantly the unchanged details are also sent to the client side and redrawn. This can be made more efficient by designing a protocol which takes into account only the changes made compared to the previous state of the game every time an event occurs.

In this implementation, the requests and responses are handled by JavaScript and HTTP. Those are Application Layer Protocols. So it introduces an overhead. But if this was implemented using Web Sockets this overhead can be reduced. Which means a faster request and response time, improving the speed of the game.