Technical Specification Document

Project 1 MPS

Multiplayer Pacman Game

1) Project Goals

This project's goal is to develop a multiplayer game similar to the Pacman Game, in which one player is the ghost (Mr. Hyde) and the rest of the players are supposed to flee from him.

2) System Architecture/Infrastructure

The chosen technology for developing the game is Unity, a cross-platform game engine, which provides many useful APIs in order to implement scalable gaming and networking features easily and quickly.

The game is based on a client-server architecture: the first player to join the game will start the server (hosted by Unity servers) and the next players will join the room created by the former.

A Unity project is composed of multiple predefined file groups. The one which was fully created by our developers is 'Assets'. In this folder, we have Art, Prefabs, Scenes and Scripts. In the first one there were defined .png files containing the images for the players – pacmans and the ghost (Mr. Hyde) as well as the image for the bricks which make up the maze. Next, we have the 'Prefabs' folder, which contains reusable objects, such as the player. It would be inefficient to have N copies of the player object, such as pacman, for N players. So, the prefab acts as a template from which you can create new object instances in the scene. The 'Assets' folder also contains the 'Scenes' folder, in which we have designed two particular ones – the Start Menu and the Game Level itself. Basically, the scenes contain multiple objects such as buttons and images for the menu and obstacles, players, camera, light for the game levels. Lastly, we have the 'Scripts' folder, in which we describe the behavioral components of our game objects. In the next section, the five main scripts of the game will be described in detail, each of them defining the logical and functional aspects of the project.

3) Implementation

The game is written in C# and it consists of five main scripts:

1) Networking
2) Scene Generation
3) Player Movement
4) Game Mechanics
5) The Score Board

3.1 Networking

This class consists of Unity specific methods to instantiate a game server. It has a unique String which is used to refer our server apart from the others Unity is hosting. So all the clients will connect using this String.

Methods:

- SpawnPlayer() – This method will create the player using the predefined asset.
- GenerateRandomPosition() – This method is used to randomly place a player on the map. It will constantly try and place the player until it is not placed on top of a brick or another player.
- RefreshHostList() – This method is used to search for servers already created by another player.
- JoinServer() – This method is used to join a previously created server (by another player).
- OnConnectedToServer() – This will call SpawnPlayer() as soon as a player joins the game.
- ResetGame() – This methods reset the players' positions and the score.
- OnGUI() – In this method we create the buttons used to create a server and to refresh the host list to view the available sessions.

3.2 Scene Generation

Our scene is based on a Boolean matrix. This means that wherever the value is set to True, there is a wall. This class consists of two methods:

- GenerateStructure() – This method is used to create C-shaped structures and will be called several times.
- Start() – This method will be called at the beginning of the game and it will start creating an outer grid (the edges of the map) and the it will repeatedly call GenerateStructure() to populate the currently empty map with randomly rotated C-shaped structures.

3.3 Player Movement

This class provides the means for a player to move around the map. It retains the player's position and direction used to check if a direction change is possible or not.

Methods:

- Start() – This method is used launch the player's movement facing right.
- Update() – This method is called once per frame and is used to update the player's current position and direction as the game advances.
- ChangeFacingFromInput() – This method is used to change the player's direction.
- SyncedMovement() – Makes sure that all the players' movement is consistent regardless of the number of frames per second.
- IsAvailableToMove() – As the name indicates, this method checks whether the player's desired direction change is possible.

- FacingToWorldScale() – This method is used to return the vector which indicates the direction that the player is facing.
- Move() – this method is called in the Update() method and is used to move the player's position as the game advances.

3.4 Game Mechanics

This class is used to implement the logic of the game: who is the ghost, who are the ones to flee from it, what happens when the ghost interacts with the rest of the players, how does the transition from being a regular player to being the ghost happen, etc.

Methods:

- OnCollisionEnter2D() – This method is used to update the ghost's score and reset the timer as soon as it collides with another player.
- changeGhost() – This method is used to transfer the ghost attribute from one player to another once the allocated time span expires. This is implemented using Unity's RPC (Remote Procedure Call). RPC calls are usually used to execute some event on all clients in the game.
- Update() – This method is called once per frame. It's usage is to determine who the ghost is at a certain point. At first, the ghost is chosen randomly from all of the connected players. After a certain time, the ghost attribute will be transferred to the player who is closest to the ghost, and the game will proceed with the new ghost.
  Moreover, this is where the interaction between the ghost and a caught player happens. If the ghost manages to capture a player, his score will increase.

3.5 Score Board

This class is where the players' scores are kept and constantly updated as the game proceeds.

Methods:

- UpdateScore() – This method will display the playerID plus his score at a certain point.
- GetScore() – This method accesses the score which is being calculated in GameMechanics and displays the value in UpdateScore().
- Update() – Called once per frame, this method calls all of the methods previously described.