Explanation of Game and Use of Object Pooling in INFR3110 Final Exam

Game info:

- WASD to move
- Collect all pellets to win
- Get hit by ghosts 3 times to lose

1.

In my game, I implemented object pooling by first creating a pool script. This script holds a list of ghost gameobjects, an integer which represents the maximum number of ghosts which can be spawned, a reference to the ghost prefab, and two floats used for spawning ghosts over an interval of time.

Since the premise behind object pooling is to create all the game objects in the pool when the program is run, instead of instantiation them when they are needed during runtime, the script's Start() function spawns in the number of ghosts equal to the maxGhosts variable and adds them to the list of game objects:

```
// Start is called before the first frame update
void Start()
{
    for (int i = 0; i < maxGhosts; i++)
    {
        GameObject newGhost = Instantiate(ghostPrefab);
        newGhost.SetActive(false);
        ghosts.Add(newGhost);
    }
}
```

Next, I created a SpawnGhost method which takes a Vector3 and a Boolean as an input. This is the method that is meant to be called in order to activate a ghost in the pool and start using it in the game. Within this method, we look through each ghost in the pool, and if we find a ghost which is not active, then we activate it, set its position to the input position, and return in order to end the method:

```
//Look through the list, find an inactive ghost
foreach(GameObject ghost in ghosts)
{
    if (ghost.activeSelf == false)
    {
        ghost.SetActive(true);
        ghost.GetComponent<Transform>().position = pos;
        return;
    }
}
```

If we go through this entire for loop and we don't find an active ghost, this means all ghosts in the pool are active and there are no more ghosts we can use. Then, we reference the input Boolean to decide if we should spawn a new ghost into the pool or not. If the Boolean is true, then we instantiate a new ghost, add 1 to the max number of ghosts, and add the ghost gameobject into our pool:

```
//If we get to this point, there were no inactive ghosts
if (overMax)
{
    maxGhosts++;
    GameObject newGhost = Instantiate(ghostPrefab);
    ghosts.Add(newGhost);
}
```

From here, we simply create a small system within the Update() function which adds time to the currentTime variable, and uses this to decide when to call SpawnGhost() based on the spawnTime variable:

```
currentTime += Time.deltaTime;

if (currentTime >= spawnTime)
{
    currentTime = 0f;

    SpawnGhost(new Vector3(0f, 1f, 0f), false);
}
```

2.

Object pooling is useful in a game like my version of Pac-Man, since it improves performance on games which rely on the re-use of the same assets. While we can just choose to instantiate ghosts when we need them, and then delete them and create new ones in the case of moving to a new level, we can instead move most of these heavy computations to when the program is first run, meaning that we get smoother performance during the actual game. This is because we instantiate the objects when the program is run, meaning that by the time the player is actually playing, all of these objects are already in the scene, and are deactivated and ready to be used. This way, the pool can simply activate these objects that have already been instantiated, and then deactivate them when they are no longer being used; this is a much lighter computation than needing to instantiate and destroy every time we want to use or remove a game object.

3.

If I were to create a management system for my game, it would be a level swapper. Pac-Man level are quite basic, especially in my version, so I would want to have multiple levels for the player to play through. To implement this, I would need to create a script which monitors the GameManager for when the game is won. Once the game is won, this script would deactivate the current level's objects, and activate the next level's objects, moving Pac-Man to this level so that he is properly positioned and ready to play it. Since the levels are so basic, it would be easier to do this as opposed to putting each level into a separate scene and loading each scene when the game is won. This manager also fits in with the use of object pooling, since by staying in the same scene, we are able to deactivate the ghosts when the player wins a level, and then proceed to spawn them in over time again for the next level.

Overall, this system would benefit the game by creating more variety for the player in a pretty easy way. It wouldn't be hard to create more basic levels for the player to play in, and this system would make it easy to simply have a list of levels that can be activated as the player progresses through the game.