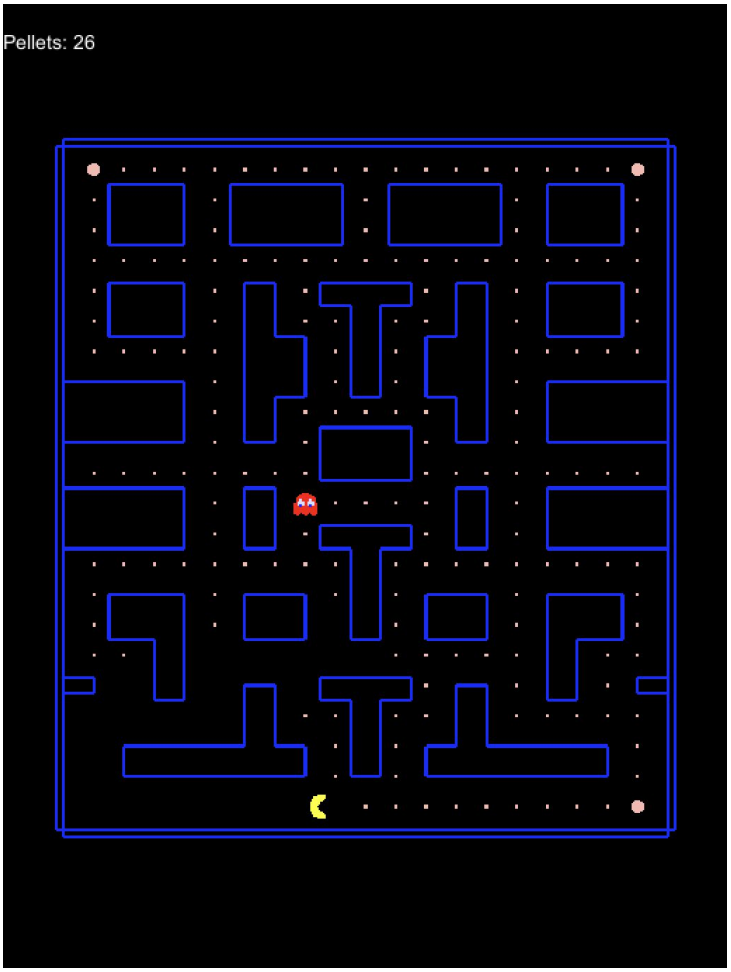
Assignment 3: Finite State Machines

The purpose of this assignment is to have you become familiar with the implementation and design of finite state machines. Finite State Machines (or FSMs) are one of the standard ways to organize and design enemy behaviors in games, breaking up enemy behaviors into particular states and the transitions between them.



In this assignment, you will implement FSMs to replicate the behavior of two of the four Pacman ghosts. You will also implement your own FSM for a new, custom ghost.

What you need to know

There are two primary scripts you will be interacting with for this assignment. In addition there are a number of other scripts and a variety of prefabs (prefabricated game components) that will be helpful for you to reference. There are a lot of moving pieces in this assignment, so it would benefit you to read this section thoroughly.

You will be required to modify the **Clyde.cs**, **Inky.cs**, and **CustomGhost.cs** scripts. All three of these scripts inherit from **FSMAgent.cs**. In addition you will need to create new states that inherit from **State.cs**

State

State is the basic state class from which all other state classes should inherit.

Member public variables:

- Name: Getter for the state's name, set in the constructor

Member functions:

- State(string _stateName): Constructor for the base state class, sets the State's name. **Note**: Ensure that any states you make have unique names
- Update(FSMAgent agent): Handles each tick of behavior for a given state, meant to be overridden. Returns the next state (return self/this if the state should remain unchanged)
- EnterState(FSMAgent agent): Set up logic upon first entering this state, meant to be overridden.
- ExitState(FSMAgent agent): Set up logic upon exiting this state, meant to be overridden.

FSMAgent.cs

FSMAgent is the basis for all of the ghosts for this assignment. You will largely be using its functions for the new states you make for this assignment

Member public variables:

- GhostAnimationHandler normal, frightened, eyes: Three ghost animation handlers for the three types of animations each ghost has
- State currState: Current state

Member functions:

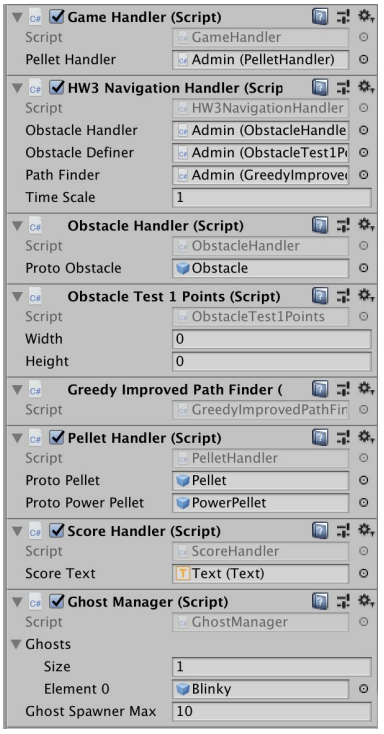
- Initiatialize: A virtual function meant to be overridden and set the initial currState.
- GetPosition: Returns the Vector3 position of this agent.
- SetTimer(float timerMax): Sets an internal timer for the given timerMax (in seconds).
- TimerComplete: Returns true if the internal timer has completed.
- SetTarget(Vector3 _target): Given a Vector3 _target begins pathing to the closest grid point to that _target.
- CloseEnough (Vector3 position): Returns true if close enough to the given Vector3 position.
- SetAnimationStateNormal(): sets the ghost to use its normal animations.

- SetAnimationStateFrightened(): sets the ghost to use its frightened animations.
- SetAnimationStateEyes(): sets the ghost to use its eyes animations.
- SetSpeedModifierHalf(): sets the ghost's speed modifier to half.
- SetSpeedModifierNormal(): sets the ghost's speed modifier to 1.0f (normal speed).
- SetSpeedModifierDouble(): sets the ghost's speed modifier to 1.5f.

In addition you might find it helpful to reference several other scripts and their functions.

- **AgentConstants.cs** (Reference via AgentConstants): Has public class variables that define Pacman's speed (SPEED), ghost's speed (GHOST_SPEED), and the ghost start position (GHOST_START_POSITION).
- **PelletHandler** (Reference via PelletHandler.Instance): Has a function GetClosestPellet(Vector3 location) that will return the closest Pellet object to the given location. Call via PelletHandler.Instance.GetClosestPellet(...).
- **GhostManager** (Reference via GhostManager.Instance) has a public variable array GhostsInPlay (reference via GhostManager.Instance.GhostsInPlay) that lists all the current ghosts in play. Also has the function GetClosestGhost(Vector3 position) (reference via GhostManager.Instance.GetClosestGhost(...)) that returns the single closest FSMAgent ghost to the given position.
- **ObstacleHandler** (Reference via ObstacleHandler.Instance): Has all the functions from assignments 1 and 2.
- **PacmanInfo** (Reference via PacmanInfo.Instance): Contains a public Facing variable that gives what direction Pacman is moving in.

Instructions



The default setup will see you controlling Pacman with clicks just as before, and a single, complete ghost (Blinky) that spawns after ~3 seconds to chase Pacman. Your responsibility for this assignment will be to finish implementing two of the ghosts (Pinky and Clyde) and create your own CustomGhost to flesh out this tiny Pacman clone. The goal for Pacman is to eat all 208 pellets in Scene1, though the game will not end when this happens. However the game will restart automatically when a ghost eats Pacman.

Step 1: Download the Assignment3.zip file from eclass, unzip it, and open it via Unity. You can open it by either selecting the unzipped folder from Unity or double clicking Scene1.

Step 2: Open Scene1 located inside Assets/Hw3. Hit the play button. Click inside the game window to see the pacman agent traverse the game world.

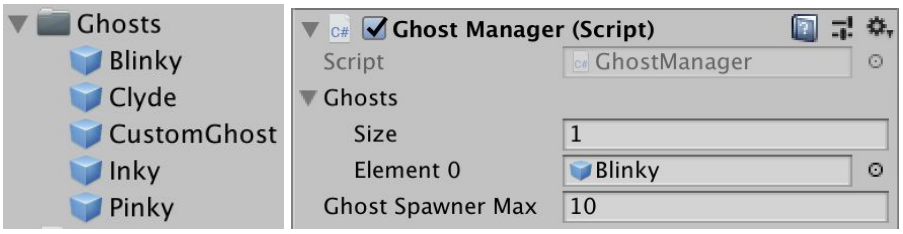
Step 2a (optional): If you wish you can replace the GridHandler.cs file provided with your GridHandler.cs file from assignment 1.

Step 2b (optional): If you wish you can make use of your A* path planner from assignment 2. To do so, copy your AStarPathFinder.cs into the HW3 folder. Then select the “Admin” object in the Scene1 Hierarchy. By default it will look like the image to the right in this section in the Inspector. Click the cog on the top right of Greedy Improved Path Finder in the Inspector and click to Remove Component. Then click and drag your AStarPathFinder.cs on top of the Admin object in the Hierarchy. Finally click and drag the A Star Path Finder bar in the Inspector into the PathFinder slot of HW3 Navigation Handler where it says “Missing (Path Finder)” (see below).

Step 3: Create the necessary State scripts and modify Pinky.cs and Clyde.cs to fully implement the correct behavior of these ghosts. Specifically, both ghosts should follow the rules of the game in terms of when they enter Scatter and Frightened states (see Blinky.cs and Inky.cs and the States they reference for examples of fully implemented ghosts). In addition **Pinky** should target the grid cell closest to four grid cells in front of Pacman, given Pacman’s current orientation. **Clyde** should target Pacman until he is within 8 cells (straight line distance) of Pacman, at which point he should target his Scatter position. **Pinky’s** Scatter position should be the top left corner and **Clyde’s** Scatter position should be the bottom left corner. For more details see: https://gameinternals.com/understanding-pac-man-ghost-behavior though you are not required to go beyond the high level description in this step.

Step 4: Create the necessary States and modify CustomGhost.cs to create your own custom ghost. This ghost should make use of at least three new and unique (no replicating the behavior of any other states) states while playing the game and should outperform all of the other four ghosts.

Step 5: Test your implementation by playing against the ghosts yourself or by using the provided AI Pacman agents to play against the ghosts. To play against one of the ghosts drag one of the ghost prefabs (items marked with a blue cube) from HW3>Ghosts> into the slot currently held by Blinky in Ghost Manager. You may also instead add extra ghosts to the Ghosts array if you wish to see how you fare against multiple ghosts.



To test one of the other Pacman agents delete “MouseClickAgent” from Scene1 in the Hierarchy view. Then drag one of the Pacman prefabs from HW3>Pacman Agents into the Scene. There are the following agents available to you:

- ArrowKeysAgent: Moves according to the arrow keys.
- MouseClickAgent: The default agent, tries to move to where you click.
- RandomAgent: Most basic AI agent, moves randomly.
- GreedyAgent: Second most basic AI agent, paths to the closest pellet.
- GreedyCowardAgent: Third most basic AI agent, paths to the closest pellet but also tries to avoid ghosts.

Grading

This homework assignment is worth 10 points. Your solution will be graded by an autograder. The point breakdown is as follows:

2 points: Correct implementation of Pinky ghost.
2 points: Correct implementation of Clyde ghost.
1 point: Custom Ghost uses at least three new states during testing. These states must be entirely new and cannot duplicate the behavior seen elsewhere in the game.
5 points: Custom Ghost outperforms the three given AI Pacman agents and two withheld agents.

You **may not** access Custom Ghost’s gameObject or transform in your code.

In addition you may receive up to 1 extra credit point if your Custom Ghost fits the game design of Pacman. In other words, while it is easy to make an overwhelming ghost, it is hard to make a ghost that presents an appropriate challenge. This will require two things. First, in addition to the three new states, your CustomGhost must also make use of the standard Scatter, Frightened, and Eyes states at the appropriate times. Second, your ghost must be harder to deal with for simpler AI agents and easier to deal with for more complex AI agents, it cannot simply immediately kill any Pacman. This will be measured by looking at the final scores of each AI agent over several runs. A good Custom Ghost will lead to the following ranking of scores of the AI Pacman agents: Random’s Score < Greedy’s Score < Greedy Coward’s Score < Withheld Agent 1’s Score < Withheld Agent 2’s Score.

Hints

Blinky alone will easily beat out any other ghost. It might be worth basing your Custom Ghost on a variation of his behavior.

You may not have your Custom Ghost teleport, but you may make use of any of the functions in FSMAgent.cs. Note that one of the functions increases the base speed.

You must have each of the three new states show up while playing against Custom Ghost. You may find it helpful to print out (with Debug.Log) what state your ghost is in.

If testing is taking too long, you may speed up the game by altering the “Time Scale” value of the HW3 Navigation Handler script on admin. Setting it to 2 will make everything two times faster. However, higher values (around 10) may end up breaking the game.

Submission

To submit your solution, upload your modified Clyde.cs, Pinky.cs, and CustomGhost.cs files. In addition you should include any new States you made. If you used your GridHandler.cs or AStarPathFinder.cs files, reupload them for this assignment.

You should not modify or upload any other files in the game engine.

DO NOT upload the entire game engine.