COMP-4770 Artificial Intelligence for Games

Overview

- **Note that Unity 2021 LTS has released and this project has been upgraded to it to take advantage of new C# features. It can not be opened with Unity 2020.**
- Since the class is now finishing, I have made the GitHub repository public here so feel free to view there.
- This project uses my own AI library, Easy AI, so there will likely be some differences between my solutions and those using Dr. Goodwin's library although Easy AI is based upon Dr. Goodwin's library. I will explain any of these differences when they occur.

- I developed Easy AI directly in this project and its fully commented source code can be found under "Packages > Easy AI".

Project

Project Getting Started

- **I made a demo video showing how to get the project open and use the GUI.**
- Under "Assets", go to "Scenes" and open "Project".
- All scripts for the project are located in "Assets > Scripts > Project".
- Clicking the "Details" button in the top left will allow for viewing team score and how the selected soldier is doing as well as messages.
- Clicking the "Controls" button in the top right will show buttons to change settings.
  - Reset: Restart the match.
  - Following Best / Manual Selection: Toggle between automatically following the best soldier and being able to click on soldiers to select them.
    - **If this is set to "Following Best", you will not be able to left click on soldiers to select them, and the "Back to Overview" button in the "Details" GUI section will not work either. You will be locked to the best soldier. Click this button to have it say "Manual Selection" if you want to be able to left click on soldiers to select them or switch soldiers through the GUI itself.**
  - Pause / Resume / Step: Pause, resume, or take a single time step.
  - Gizmos: Switch between showing gizmos for the current soldier, all soldiers, or none to see which soldiers are looking where.
  - Nodes: Switch between showing all navigation paths in the level, those being used by all soldiers, those being used only by the selected soldier, or none.
  - Buttons to switch between all cameras.
  - The scenes buttons are not relevant for the project, these will take you between the scenes for the assignments.

Project Requirements

1. **Document your work. Explain what parts you attempted, how your approach is supposed to work, which parts of the code you modified, removed or added.**

- Every method is fully commented, including the hundreds of lines for the project (and all of the assignments), and the thousands of lines I've written when creating Easy AI. I'm hoping all of this

documentation will make it easy to port of specific features that could be useful in future versions of libraries for COMP-3770 and COMP-4770.

- Unlike when doing assignments, where I would first look at the provided template to base my own implementation off of, I challenged myself to do the project based solely off the outline and did not look at the provided template, so my implementation is going to be very different from the template.

2. **Tune the parameters and evaluators. Some parameters in Parameters.cs are not used or not tuned. You can remove unused parameters (or make use of them). You should also adjust the values of the parameters to be sensible. For example, what should the sound range for the shotgun be? What should the rate of fire be? How many searches should be allowed per update? Document your changes and rationale.**

- Given my approach was completely from scratch, parameters are different, so I will outline those which I have created below.
    - Core Parameters on the "Soldier Agent Manager":
        - **Soldiers Per Team**: How many soldiers to have on each team which can be between 1 and 15. My computer was easily capable of running 15, but I didn't bother trying more as the map was packed with 30 total soldiers as was. Set to 15.
        - **Health**: How much health soldiers have. Set to 100.
        - **Respawn**: The respawn time in seconds. Set to 10 seconds.
        - **Pickup Timer**: How long pickups are disabled for after being picked up. Set to 10 seconds.
        - **Memory Time**: How long enemies that soldiers see or hear will remain in memory. Set to 5 seconds.
        - **Low Health**: The health limit of which below a soldier will consider itself as being at low health. Set to 50.
        - **Max Wait Time**: After reaching a destination, this is the maximum time in seconds a soldier will stay in the same place before deciding on a new place to move to. Set to 5 seconds.
        - **Distance Close**: If an enemy is closer than this many units it will be considered close. Set to 10 units.
        - **Distance Far**: If an enemy is farther than this many units it will be considered far. Set to 20 units.
        - **Volume**: How loud the sounds are. *This has no impact on soldiers, they will still "hear" shots themselves, this is simply for your own audio level.* Set to 1%.
    - Weapons Parameters:
        - All Weapons Parameters:
            - **Max Ammo**: How much ammo a soldier can carry for this weapon.
                - Machine Gun: 30
                - Shotgun: 10

        - Sniper: 5
        - Rocket Launcher: 1
        - Pistol: Unlimited
    - **Damage**: How much damage a shot from the weapon does.
        - Machine Gun: 10
        - Shotgun: 30
        - Sniper: 100
        - Rocket Launcher: 200
        - Pistol: 10
    - **Delay**: How much time in seconds between shots.
        - Machine Gun: 0.1
        - Shotgun: 1
        - Sniper: 1.5
        - Rocket Launcher: 3
        - Pistol: 0.25
    - **Time**: How long the bullet trail (for raycast/hit scan weapons) or projectile (for projectile weapons) lasts.
        - Machine Gun: 0.1
        - Shotgun: 0.1
        - Sniper: 1.5
        - Rocket Launcher: 30
        - Pistol: 0.1
    - **Rotation Speed**: How quick in degrees per second a soldier can rotate when using a certain weapon.
        - Machine Gun: 15
        - Shotgun: 30
        - Sniper: 3
        - Rocket Launcher: 3
        - Pistol: 15
    - **Sound Range**: How far away other soldiers can hear a shot from this weapon in units.
        - Machine Gun: 20
        - Shotgun: 20
        - Sniper: 1000
        - Rocket Launcher: 1000
        - Pistol: 10
- Raycast/Hit Scan Weapons Parameters:

        - **Rounds**: How many rounds to fire during a single shot.
            - Machine Gun: 1
            - Shotgun: 10
            - Sniper: 1
            - Pistol: 1
        - **Spread**: How much random spread to add to bullets.
            - Machine Gun: 0.015
            - Shotgun: 0.085
            - Sniper: 0
            - Pistol: 0.015
    - Projectile Weapons Parameters:
        - **Velocity**: How fast projectiles travel in units per second.
            - Rocket Launcher: 50
        - **Distance**: How far splash damage from projectiles can be applied.
            - Rocket Launcher: 10

3. **Design and implement an additional goal-oriented behaviour and a corresponding evaluator and relevant features. For example, MoveToCover or CaptureTheFlag. Document what you did and how it works.**

- **All soldier logic can be found in "Assets > Scripts > Project > Agents > SoldierAgent.cs" where all code is fully documented to go into even more detail than this.**
    - At first glance, this method may not directly seem like goal-oriented behaviour design in terms of how the provided library or Buckland's book outlines it, but, this is just a result how I designed Easy AI over this semester as I will explain:
        - I designed Easy AI with the idea of it being very approachable to someone with limited Unity experience and thus classes are larger and not as component based as I would normally do with Unity.
            - On this topic, one of the greatest difficulties other students in the class that would ask me for help would tend to have was they tended to get lost or overwhelmed with how modular and components based the templates were. I would explain to them that modular components are a best practice for Unity, but, given most students only have Unity experience from COMP-3770 and nothing else, trying to understand all components of such a large library seemed to overwhelm them at times. Thus, why I took a completely opposite approach with Easy AI to limit the number of components needed, which could be something to consider with developing class libraries for future COMP-3770 and COMP-4770 classes (perhaps more so for COMP-4770 where the focus isn't necessarily Unity specific, and keep more modular components for COMP-3770 to teach students best Unity practices).

        - Since I built up functionality over each assignment, the "goals" are often all part of the same class but how the behaviours are built on top of each other can still be seen:
            - The soldiers make their top-level decisions taking into account what their role on the team is, if they or the enemy has the flag, how much health they have, what weapons have ammo, and how far the enemy they are fighting is or if they even see or hear any enemies to begin with. These then will decide where the enemy will move to which is done with pathfinding.
            - A path is then determined and the soldier's base agent class starts the behaviour of following the path.
            - This pathfinding behaviour then calls the steer behaviour towards the next point in the path.
            - Upon being close enough to the point in the path, the agent will remove it from the path to begin seeking towards the next point.
- My soldiers are capable of collecting and capturing the enemy flag as well as hunting down an enemy which is carrying the flag and then moving to return their flag.
    - Each team designates a soldier as the collector which will with a path to the enemy flag and once having it will find a path back to their base to capture it.
    - Each team has designated defenders that will move to kill an enemy which has taken the flag and return it to their base.
- Agents can move to designated offensive or defensive positions.
    - Attacker soldiers will move between offensive positions.
    - Defender soldiers will move between defensive positions.
- Agents can move to pickup health and ammo picks.
- Agents will pick an ideal weapon to use based on whether or not they are an attacker or defender, how far away they are from the enemy they are engaging, and what weapons they have ammo in.

4. **Improve the game. Here are some ideas.**

    i. **BestPathTable.cs contains a pre-computed path table similar to the ones discussed in the lectures. Modify the PathPlanner and/or PathManager to use this table instead of A Star Search and Dijkstra's Search. That should improve the efficiency of the game.**

        a. I have made a custom implementation for this which I did during Assignment 4.

        b. The lookup data for the project can be found in the root of the Unity project and going to "Navigation > Project.txt".

        c. For more details on how this was implemented, go to the end of the A4 Requirements section where it is fully explained.

    ii. **Implement the ability to pickup the enemy flag at the base and carry it back to your base. If you are killed while carrying the flag, simply return the flag to its base or implement a means**

for the flag to be dropped where you died so either a team-mate or the enemy can pick it up. This may require implementing a new form of trigger.

   a. Flags can be picked up, captured, dropped when a carry is killed, and retrieved to be returned to base.
   b. The code for the flags can be found in "Assets > Scripts > Project > Pickups > FlagPickup.cs".

iii. **Improve issues with the frame rate so that the game plays smoothly with 3, 4 or 5 weebles per team.**

   a. My implementation has been able to run without any issues at very high frame rates with thirty soldiers on the field, and I simply did not try with more because I was running out of map space.

iv. **Improve the various message displays to provide even more useful info.**

   a. Although my GUI implementation has both pros and cons to those in the provided library, I like it as it is fully automatically generated without the need to add any additional objects or components into the scene making it great for testing purposes, especially for this class where we are more focused on the logic behind agents in games rather than the visuals and UI of a game.

v. **Provide additional buttons or key inputs to control various functionality.**

   a. Buttons to easily expand or collapse the entire GUI.
   b. Button to reset the level.
   c. Button to toggle between following the best soldier or to allow for manual selection.
   d. Buttons to toggle all gizmos for where soldiers are looking and what paths are shown.
   e. Buttons to switch between all cameras.
   f. Left clicking on a soldier will switch to following that agent **provided you have it set to manual selection and not automatically following the best soldier.**

vi. **Add a team score (currently, only individual scores are recorded).**

   a. The number of flag captures and kills each team has gotten are tracked and display in the GUI.

vii. **Add ability to minimize and resize message windows (current only draggable).**

   a. As said above, there are buttons to easily expand or collapse the entire GUI.

viii. **Add ability to open a weeble's EntityConsole by clicking on the weeble.**

   a. As said above, left clicking on a soldier will switch to following that agent **provided you have it set to manual selection and not automatically following the best soldier.**

ix. **Design and implement even more goal-oriented behaviours and a corresponding evaluators and relevant features. Make use of the low walls and elevated areas for cover and good sniping locations.**

   a. These were detailed earlier with agents being able to move into advantageous offensive and defensive positions.

x. **Find and document bugs. Suggest design improvement. Suggest new features.**
    a. Everything is fully documented, and all features are welcome to be added into future libraries for COMP-3770 and COMP-4770. As I said above, one of the greatest difficulties other students in the class that would ask me for help would tend to have was they tended to get lost or overwhelmed with how modular and components based the templates were. I would explain to them that modular components are a best practice for Unity, but, given most students only have Unity experience from COMP-3770 and nothing else, trying to understand all components of such a large library seemed to overwhelm them at times. Thus, why I took a completely opposite approach with Easy AI to limit the number of components needed, which could be something to consider with developing class libraries for future COMP-3770 and COMP-4770 classes (perhaps more so for COMP-4770 where the focus isn't necessarily Unity specific, and keep more modular components for COMP-3770 to teach students best Unity practices).

xi. **Go nuts. Add sound effects, explosions, etc. Make your game cool.**
    a. Made a castle-style level while keeping the same general shape of the original level.
    b. Added in weapon models.
    c. Added in shoot and impact sound effects.
    d. Bullet impact and rocket explosion effects.

5. **Have fun!**

- This has been my favourite class out of all my years at the University of Windsor, definitely had a lot of fun with this course!

Assignment 4

A4 Getting Started

- Under "Assets", go to "Scenes" and open "Assignment 4".
- **When running the scene, right click anywhere on the level with the mouse and the agent will find a path to that location.**
- The general controls for the scene are as follows:
    - Click the "Controls" button to see:
        - Buttons to pause, resume, or step through the scene.
        - Buttons to switch between cameras.
        - **Click the button which has "Nodes" in its text to change what nodes are drawn.**
- **Unlike Dr. Goodwin's starter project, my equivalent to "MapData.txt" is located in the root of the project in the "Maps" folder and is named "Assignment 4.txt". Note that I did not add in the**

numbers on the sides of the text file as I did not need them for my implementation since my implementation does not actually use this text file directly and I am simply outputting it for review purposes only.

- **My node generation graphs generates a lookup table with A* which allows for quick loading in from a text file as opposed to a lengthy generation period at each run. By default I have this enabled and details as to how to toggle this will be in the requirements section below.**
- I have copied over the core level geometry from Dr. Goodwin's sample level for the assignment into my project but applied a matte material style to it.

A4 Requirements

1. **Familiarize yourself with the provided framework and the code reviews in class.**

- Read through all node and navigation related classes which served as the base for my implementation in Easy AI.

2. **Explore the provided scripts in GridSearchSpace.cs, LevelInfo.cs, and A4Testing.cs.**

- Recreated my own version of grid search space which can be found in "Packages > Easy AI > Runtime > GridGenerator.cs".
  - Note that this is not currently in use in the scene as the corner graph is used instead.
- Recreated and improved "LevelInfo.cs" where its equivalent in Easy AI can be found at "Packages > Easy AI > Runtime > NodeArea.cs" which does not have hardcoded level sizes and can instead generate nodes for any size of level.
  - More info on this at the end of this requirements section.

3. **Use ideas from GridSearchSpace.cs to complete the implementation of CornerGraphSearchSpace.cs. You will need to test for corners using LevelInfo.cs and its provided methods as well as any others you need. For convex corner detection, see example below.**

- The equivalent of "CornerGraphSearchSpace.cs" in Easy AI can be found at "Packages > Easy AI > Runtime > CornerGraphGenerator.cs".
- This class is fully commented with the main node placing logic in the "Generate" method.

4. **Test and document.**

- Every method and every parameter in every class is fully documented.
- Testing is extremely easy as all that is needed it to right click with your mouse anywhere in the level and the agent will navigate to that position.

5. **Go above and beyond to distinguish yourself. Remember, 20% of the marking is competitive.**

- Complete lookup table generation, saving, and loading.
    i. "NodeArea.cs" found at "Packages > Easy AI > Runtime > NodeArea.cs" generates nodes based off an attached generator such as "CornerGraphGenerator.cs" or "GridGenerator.cs".
    ii. Nodes are then connected in "NodeArea.cs" based off of line of sight, taking into account the radius of agents to ensure no paths that are too thin are made. **Note that because of this, some connections seen in the assignment handout such as the ones on the far left and right of the images where a line is going over an obstacle are not made as those do not take into account the agent radius.**
    iii. After all nodes and connections are generated, A* pathfinding is done between all nodes.
        a. The A* method can be found in the "AgentManager.cs" class at "Packages > Easy AI > Runtime > AgentManager.cs" in the method named "AStar".
    iv. All moves from each node to another are stored in memory for easy lookup and fast navigation as well as saved to a file.
        a. **The lookup table is saved in the root of the project in the "Navigation" folder in "Assignment 4.txt".**
    v. **On the "AgentManager" in the scene, enabling the parameter "Lookup Table" will have future runs load this data back from the text file instead of going through the above steps again, greatly reducing startup times. Due to this greatly reduced time, I have enabled this by default for your convenience. If you wish to run the generation yourself such as if you change the level geometry or simply want to try it out, uncheck this field but be aware that due to needing to perform every A* combination in the level this will take some time and Unity will be unresponsive for a while once entering play mode.**
        a. The methods for reading and writing this data can be found in the "AgentManager.cs" class at "Packages > Easy AI > Runtime > AgentManager.cs" in the "WriteLookupData" and "ReadLookupData" methods.
- String pulling for paths.
    - Paths are read from the lookup table in the "AgentManager.cs" class at "Packages > Easy AI > Runtime > AgentManager.cs" in the "LookupPath" method where nodes will be skipped if the agent has a direct path from them to another node or its destination.
- Click to move.
    - Right click with your mouse anywhere in the level to have the agent navigate to that position.

Assignment 3

A3 Getting Started

- **The main script for this assignment which contains all the steering behaviours has been build into my "Easy AI" package and can be found at "Packages > Easy AI > Runtime > Steering.cs".**

    - Unlike Dr. Goodwin's steering behaviours, I have made mine as static functions in this single file so they are not tied directly to agents/actuators/regulators as they are in the GameBrains library so they could easily be implemented by other types of behaviours and objects.
    - Thus, for agents implementing the movement, this is done in the "Agent.cs" script at "Packages > Easy AI > Runtime > Agent.cs" in the "CalculateMoveVelocity" and "Look" methods.
- Under "Assets", go to "Scenes" and open "Assignment 3".
- The general controls for the scene are as follows:
    - Click the "Details" button to see:
        - Buttons to select either the red or blue agent.
        - Once an agent is selected, buttons to perform various seek, pursue, flee, evade, and wander actions are available for you to click.
        - To get back to the agents list to select a different agent, click the "Back to Overview" button.
    - Click the "Controls" button to see:
        - Buttons to pause, resume, or step through the scene.
        - Buttons to switch between cameras.
- **Assignment 1 and 2 have both been overhauled to use steering behaviours as part of the "distinguish yourself" part of this assignment which can be accessed by opening the "Assignment 1" and "Assignment 2" scenes with additional documentation on those parts available in their own sections in this readme file.**

A3 Requirements

1. **Familiarize yourself with the provided framework. The key script to understand is Seek.cs, which implements the Seek steering behaviour. Observe how Steer() computes a steering output.**

- Reviewed Dr. Goodwin's code as well as Buckland's book to develop my own seek behaviour which can be found in "Packages > Easy AI > Runtime > Steering.cs".

2. **Implement the Flee steering behaviour. You can do this by completing the Steer() method in Flee.cs. This version of Flee inherits from LinearSlow like Seek does and is a straightforward modification of Seek.cs. An alternative is to inherit from Seek or contain an instance of Seek and let Seek do the heavy lifting rather than replicate code from Seek in Flee.**

- Implemented the flee steering behaviour based on Buckland's book which can be found in "Packages > Easy AI > Runtime > Steering.cs".

3. **Implement the Pursuit behaviour by completing the Steer() method in Pursue.cs. For this you will need to compute the predicted future position of the target.**

- Implemented the pursuit steering behaviour based on Buckland's book which can be found in "Packages > Easy AI > Runtime > Steering.cs".
- *To best see the difference between seek and pursuit, in the "Assignment 3" scene, observe the differences in the path an agent takes when clicking the "Seek Spinner" and "Pursue Spinner" buttons where the pursuit behaviour will aim ahead of the target to intercept it.*

4. **Implement Evade by completing the Steer() method in Evade.cs. Just as Flee is a mirror of Seek, Evade is a mirror of Pursue.**

- Implemented the evade steering behaviour based on Buckland's book which can be found in "Packages > Easy AI > Runtime > Steering.cs".
- *To best see the difference between flee and evade, in the "Assignment 3" scene, observe the differences in the path an agent takes when clicking the "Flee Spinner" and "Evade Spinner" buttons where by looking at the ground below the agent, you can notice in the evade behaviour it is influenced by the velocity of the spinning cube as the agent does not move in a straight line.*

5. **Complete the Steer() method of at least one other steering behaviour in Steering/VelocityBased.**

- I completed two additional steering behaviours being "Wander" and "Face" which can be found in "Packages > Easy AI > Runtime > Steering.cs".
- "Wander" was based off of the behaviour outlined in Dr. Goodwin's slides where he using a random binomial value to steer the agent.
- "Face" was a custom implementation of my own.

6. **Distinguish yourself by completing more of the steering method templates, or adding additional steering behaviours beyond the given templates, or adding acceleration-based steering behaviours, or completing the prioritized or dithering combining methods in SteeringData. Be creative and have fun.**

- As stated above, I completed two additional steering methods over the required one.
- Agents can operate by either velocity based or acceleration based movement using my steering behaviours.
    - By default, in all scenes, all agents are already using acceleration based movement where they must accelerate to a max speed and decelerate back to zero when stopping.
    - To switch to velocity based movement, simply change the "Move Acceleration" field on an agent (or its prefabs prior to running the scene) to be zero.
- **Assignment 1 and 2 have both been overhauled to use steering behaviours as part of the "distinguish yourself" part of this assignment which can be accessed by opening the "Assignment 1" and "Assignment 2" scenes with additional documentation on those parts available in their own sections in this readme file.**
    - In assignment 1, the change is to have the agent seek to dirty floor tiles.

- In assignment 2, more drastic changes have been made:
  - Instead of a sleeping state, there is now a wandering state where microbes will wander randomly.
  - Microbes now pursue the microbe they want to eat and that microbe attempts to evade it.
  - Microbes that wish to mate pursue each other.
  - Microbes seek pickups.

Assignment 2

A2 Getting Started

- Under "Assets", go to "Scenes" and open "Assignment 2". The level will generate itself and spawn the cleaner agent when you click play.
- All scripts for assignment one are located under "Assets > Scripts > A2" and the sub folders within.
- The prefabs are all under "Assets > Prefabs > A2".
- The general flow of the scene is as follow:
  i. A round floor is generated. I choose a round floor to make it seem like a petri dish.
  ii. An initial population of microbes is spawned along with pickups.
  iii. From there, microbes interact solely based upon their states with the scene only spawning in more pickups as needed and more microbes if the population drops too low. The general flow that agents take from birth to death are as follows:
    a. Start in a wandering state as an infant.
    b. If at any point the microbe becomes hungry, it will search for food.
    c. If the microbe is an adult and is not hungry and it has yet to mate, it will search for a mate.
    d. If the microbe is an adult and is not hungry and it has already mated, it will search for a pickup which depending on the result of the pickup could cause it to want to mate again.
- The general controls for the scene are as follows:
  - Click the "Details" button to see:
    - A list of the microbes.
    - Further clicking buttons in this GUI will allow you to view messages for specific microbes and see their states, hunger, lifespan, etc.
      - **Note that if the "Oldest Camera" is selected, you cannot select specific microbes as it will constantly reset itself back to viewing the details of the oldest microbe so you will need to switch to a different camera. See the next few lines on "Controls" below.**
  - Click the "Controls" button to see:
    - Button to reset the scene.

        - Buttons to pause, resume, or step through the scene.
        - Buttons to switch between cameras.
- **I have a relatively strong computer so I've been able to get the microbe count pretty high without issues, however, if you are having performance issues, in the scene select "Microbe Manager" and reduce the "Min Microbes" and "Max Microbes" fields.**

A2 General Details

- In my library, the equivalent features of Dr. Goodwin's StateMachine class has been built into the Agent class to merge the two but behaves essentially identically.
    - This can be found in "Packages > Easy AI > Runtime > Agent.cs".
- In my library, the equivalent features of Dr. Goodwin's StateManager class has been built into the AgentManager class to merge the two but behaves essentially identically.
    - This can be found in "Packages > Easy AI > Runtime > AgentManager.cs".
- The optional task of creating a camera that follows the oldest microbe and can zoom in and out has been completed.
- The main difference between mine and Dr. Goodwin's library is how events are passed between agents. The AIEvent class located at "Packages > Easy AI > Runtime > AIEvent.cs" is a wrapper class for sending events with relevant data to other agents. It works by having a unique integer ID field which for this assignment is from an enumeration in the MicrobeManager class which is located at "Assets > Scripts > A2 > Managers > MicrobeManager.cs" that allows a receiving agent to identify what the message type is, the agent which sent the message, as well as an optional data object. All this allows for easily and efficiently passing data between agents which in the case of this assignment are the microbes.

A2 Microbes

- I have increased the types of microbes up to seven being red, orange, yellow, green, blue, purple, and pink microbes.
- Microbes can mate with and eat various colors as seen in the table below where an "M" means those microbes can mate and a blank space means those microbes can eat each other:

	Red	Orange	Yellow	Green	Blue	Purple	Pink
Red	M	M					M
Orange	M	M	M				
Yellow		M	M	M			

	Red	Orange	Yellow	Green	Blue	Purple	Pink
Green			M	M	M		
Blue				M	M	M	
Purple					M	M	M
Pink	M					M	M

- Microbes have a set lifetime which upon being reached they will die. If a microbe has reached or exceed half its lifetime it is declared an adult, otherwise, it is an infant. Only adults can mate.
- Microbes increase in size as they age.
- Microbes can only mate once, although a pickup can allow them to mate again.
    - Microbes inherit from their parents meaning the microbe will have the color of one of its parents and its other attributes such as lifespan, speed, and detection range will be the average of its parents plus or minus a slight random variation.
- Microbes display unique particles and play unique sounds when spawning, eating, mating, and picking up pickups.

A2 States

- States can be found in "Assets > Scripts > A2 > States".
- As seen in the comments of Dr. Goodwin's code, it was suggested we experiment with removing the dead state which is what I have done. This state is not needed if instead the microbe is simply removed when it is killed instead of needed to switch states.
- Further optimizations I made included merging the mating and reproducing states into one being the "MicrobeSeekingMateState".
- An additional state I introduced was "MicrobeSeekingPickupState" which will search for pickups.
- States are visually identifiable by the color of the "hats" on the microbes.
    - Black means wandering.
    - Grey means searching for food.
    - White means searching for a mate.
    - Shiny white means searching for a pickup.

A2 Pickups

- I have added four bonus pickups for microbes to search for and pickup. These are located in "Assets > Scripts > A2 > Pickups".

1. FertilityPickup allows for the microbe to mate again.
2. NeverHungryPickup sets the microbe's hunger to the lowest possible negative integer value which for all intents and purposes means the microbe will never be hungry in its lifetime again.
3. OffspringPickup will spawn several offspring right away without needing a mate.
4. RejuvenatePickup will cause the microbe to revert its age back to when it just became an adult, meaning in theory a microbe can get lucky and repeatedly pick these up to live forever.

Assignment 1

A1 Getting Started

- Under "Assets", go to "Scenes" and open "Assignment 1". The level will generate itself and spawn the cleaner agent when you click play.
- All scripts for assignment one are located under "Assets > Scripts > A1" and the sub folders within.
- The prefab for the cleaner agent is located at "Assets > Prefabs > Cleaner Agent".
- The general flow of the scene is as follow:
    i. A floor is generated of multiple tiles each with a "Floor" component attached to them where a percentage of floor tiles are twice as likely to get dirty as others being distinguished by their whiter and shinier material.
    ii. At a set interval, tiles are randomly chosen to increase in dirt level which changes their materials.
    iii. If the agent is on a dirty floor tile, it cleans it which takes a set amount of time. Otherwise, the agent moves and looks towards the nearest dirty floor tile if there are any. If all tiles are clean, the agent calculates and moves towards the weighted midpoint of the level being the optimal place to wait while factoring in the tiles that are more likely to get dirty than others.
- The general controls for the scene are as follows:
    - Click the "Details" button to see:
        - The agent, its performance, position, rotation, and all messages.
        - Further clicking buttons in this GUI will allow you to view messages for specific sensors and actuators.
    - Click the "Controls" button to see:
        - Buttons to increase or decrease the size of the floor.
        - Buttons to pause, resume, or step through the scene.
        - Buttons to switch between cameras.

A1 Requirements

1. **Review the A1 project files (get A1-Agent branch from GitLab then add to Unity).**

- Went through the core of the library to help build the foundation of Easy AI.

2. **Consider the design of the Agent Architecture. It is a first cut at providing a somewhat general framework that could be part of a Game AI library.**

- Same as above, went through the core of the library to help build the foundation of Easy AI.

3. **You may revise and/or extend it as you see fit. Bear in mind that it should be something you can build on and handle a variety of agent types.**

- Did exactly this and remade my own library Easy AI to ensure I had a full understanding of the concepts for a general agent while also making some design changes that fit better with my general workflow with Unity projects.

4. **Search the source files for "// TODO for A1" comments. This will direct you to points in the code where changes are need.**

- These comments were related to filling in the gaps for getting the agent to work for assignment one which involved allowing the mind the think, agent movement and turning along with optional acceleration, and the sensors and actuators which needed to be completed.
- These issues are all taken care of in my solution, even though the logic is in some areas in different places, as my cleaner agent thinks based upon two sensors for the floor, moves and turns with acceleration rates, and cleans through an actuator which takes a set amount of time to finish cleaning.

5. **Create a test environment for a vacuum agent. This should have at least five locations with the same Y and Z (or X and Y) coordinates. There should be a way to visually indicate, sense, and change the clean/dirty status of a location. One approach might be to have each location be a separate plane and use its colour to indicate its status. You may use the provided environment or create your own fanciful world.**

- The environment is procedurally generated by setting the "Floor Size" on the "Cleaner Agent Manager" which can create any environment size from a 1x1 space to in theory as large as you want. By default I have it generating a 5x5 floor area so 25 floor tiles.
    - This can be adjusted at runtime as well by clicking the "Controls" button and then the increase and decrease X and Y buttons. For simplicity these runtime controls allow for a max size of 5x5 so the cameras fit the whole scene.
- Floor tiles can be either clean, dirty, very dirty, or extremely dirty which will change their material depending upon their state so they are visually identifiable.
    - Clean floor tiles are a grey color. If a tile is clean and it is also "likely to get dirty", meaning it has double the chance of normal floor tiles to increase in dirty level every time dirt is added to the

floor, its clean material is instead a shiny white. More on this later in requirement thirteen.
    - Dirty floor tiles are a tan color.
    - Very dirty floor tiles are a light brown color.
    - Extremely dirty floor tiles are a dark brown color.
- At least one floor tile is increased in dirt level at a set time interval.
    - This is controlled by the field "Time Between Dirt Generation" field on the "Cleaner Agent Manager" which I have set to five seconds.
- The chance a floor tile gains a dirt level during these updates is given by "Chance Dirty" on the "Cleaner Agent Manager" which I have set to five percent. A random number between zero and one is generated three times for each floor tile and if the value is less than or equal to this percentage, the dirt level increases, meaning there is a chance for a floor tile to gain multiple dirt levels during a single generation.
- "Likely To Get Dirty Chance" on the "Cleaner Agent Manager" is used during initial floor generation and determines the odds a floor tile is twice as likely to get dirty compared to others giving it the shiny white material when clean over the standard gray material. More on this later in requirement thirteen.

6. **Add a SimpleReflexMind for the vacuum agent using SeekerMind for inspiration.**

- The mind is called "CleanerMind" instead of "SimpleReflexMind" and it is located in "Assets > Scripts > A1 > Minds".
- The mind is attached to the cleaner agent.
- The mind operates by first seeing if its sensors detected if it is standing on a dirty floor tile in which case it stops to clean the current floor tile. Otherwise, it moves towards the nearest dirty floor tile.

7. **Add a sensor to detect the location and status of the agent's current position.**

- The cleaner agent has two sensors both of which the scripts for can be found in "Assets > Scripts > A1 > Sensors" and both are attached to the cleaner agent.
- "FloorsSensor" reads all floors in the level and returns their position and if they are dirty to the agent in a "FloorsPercept".
- "DirtySensor" detects if the floor tile the agent is currently on is dirty or not and returns this to the agent in a "DirtyPercept".

8. **Add a suction actuator to change the dirty/clean status of the location of the agent.**

- The actuator is named "CleanActuator" and is located in "Assets > Scripts > A1 > Actuators" and is attached to the cleaner agent.
- If it receives a "CleanAction" from the agent, it will clean the floor tile the agent is currently on.

- It's "Time To Clean" field indicates the time in seconds it takes to clean a floor tile by one dirt level which I have this set to 0.25 seconds. This means that if the agent wishes to clean an extremely dirty floor tile, it will take them 0.75 seconds to make the tile fully clean.

9. **Add an appropriate performance measure.**

- The performance measure is named "CleanerPerformance" and is located in "Assets > Scripts > A1 > PerformanceMeasures" and is attached to the cleaner agent.
- The performance measure is a value from zero to a hundred representing a percentage as to how clean the entirety of the floor is. If the entire floor is clean, it is a hundred percent. If the entire floor is extremely dirty, it is zero percent.
- The current value of the performance measure can be seen by clicking the "Details" button when the scene is running.
- The agent in this scenario has no concept of energy level thus it never gets tired or experience any similar effects and its sole objective is to keep the floors clean, thus, I did not add any factor to the performance measure involving how much the agent moves since there realistically is no punishment for this. If this was a more advanced multi-agent scenario where perhaps after an agent got tired and needed to rest it would swap roles with another agent, then adding in a measure for movement would be valuable, but in this limited scenario, it serves little benefit thus I did not incorporate one.

10. **Add anything else you need for functionality, elegant framework, creativity, esthetics.**

- Developed the Easy AI framework along with its custom GUI.
- Multiple cameras which can all be switched to by hitting the "Controls" button when the scene is running.
- Resizeable floor by hitting the "Controls" button when the scene is running.
- Particle effects for the vacuum cleaner model when it is cleaning.

11. **Review the lectures where additional details will be provided including a walk through of the library.**

- As stated earlier, went through the core of the library to help build the foundation of Easy AI and attended all lectures.

12. **See the video of my 80% solution.**

- Able to do the required 5x1 floor size as seen in the video and any other floor size due to its procedural generation.
- Floor tiles change color based on their state as seen in the video with mine having additional dirty levels.
- Multiple camera perspectives which can be controlled from the GUI as seen in the video.

- Messages displayed to a console window as seen in the video which can be seen by clicking the "Details" button.

13. **Challenge: What if the locations had different probabilities of becoming dirty and the agent knew this? Could its performance be enhanced? Example: Think of strategies for positioning the read/write head of a disk or the idle position of an elevator based on anticipated requests.**

- Floor tiles which are white and shiny are "likely to get dirty" meaning they are twice as likely to get dirty as other tiles when dirt is added to the floors.
- For instance, if floor tiles have a five percent chance to get dirty, these "likely to get dirty" floor tiles have a ten percent chance of getting dirty.
- If the entire floor is clean, instead of returning to the exact center of the floor, the agent calculates the weighted midpoint of all floor tiles. You can see the agent's current position at any point in time by clicking the "Details" button.
    - First, it sums the positions of all floor tiles once.
    - Then, it adds the positions of all floor tiles that are likely to get dirty again which will shift this midpoint as now these tiles have been added to the sum twice given they are twice as likely to get dirty.
    - Then, the agent moves to this location. Given the nature of this being a square floor, this will still be close to the center of the floor, and depending upon how the floor was randomly generated, may be the exact center, however, it is often slightly off center to be at the weighted center of the floor tiles.

Assets

Project Assets

- Environment - Retro Medieval Kit | Kenney
- Weapons - Blaster Kit | Kenney
- Effects - War FX | Jean Moreno (JMO)
- Sounds - Free Sound Effects Pack | Olivier Girardot

Assignment 2 Assets

- Sounds - Sci-Fi Sounds | Kenney

Details

- Created using Unity 2021.3.1f1.
- Created using Easy AI.
- Created using the Universal Render Pipeline.
- Created using Unity's Input System.