COMP 4770 - Artificial Intelligence for Games

Overview

- If you would like access to the GitHub repository, email me your GitHub account and I will add you to the repository here which otherwise I am keeping private so others in the class do not have access to my solutions.
- 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 have shared this library with Dr. Goodwin since the first assignment and he was okay with me building and using my own.**
  - I developed Easy AI directly in this project and its fully commented source code can be found under "Packages > Easy AI".

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".

```
/// <summary>
/// Seek - Move directly towards a position.
/// Based upon the implementation detailed in Programming Game AI by Example page 91.
/// </summary>
/// <param name="position">The position of the agent.</param>
/// <param name="velocity">The current velocity of the agent.</param>
/// <param name="evader">The position of the target to seek to.</param>
/// <param name="speed">The speed at which the agent can move.</param>
/// <returns>The velocity to apply to the agent to perform the seek.</returns>
public static Vector2 Seek(Vector2 position, Vector2 velocity, Vector2 evader, float speed)
```

```
{
    return (evader - position).normalized * speed - velocity;
}
```

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".

```
/// <summary>
/// Flee - Move directly away from a position.
/// Based upon the implementation detailed in Programming Game AI by Example page 92.
/// </summary>
/// <param name="position">The position of the agent.</param>
/// <param name="velocity">The current velocity of the agent.</param>
/// <param name="pursuer">The position of the target to flee from.</param>
/// <param name="speed">The speed at which the agent can move.</param>
/// <returns>The velocity to apply to the agent to perform the flee.</returns>
public static Vector2 Flee(Vector2 position, Vector2 velocity, Vector2 pursuer, float speed)
{
    // Flee is almost identical to seek except the initial subtraction of
    // positions is reversed.
    return (position - pursuer).normalized * speed - velocity;
}
```

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.*

```
/// <summary>
/// Pursuit - Move towards a position factoring in its current speed
/// to predict where it is moving.
/// Based upon the implementation detailed in Programming Game AI by Example page 94.
/// </summary>
/// <param name="position">The position of the agent.</param>
/// <param name="velocity">The current velocity of the agent.</param>
```

```
/// <param name="evader">The position of the target to pursuit to.</param>
/// <param name="targetLastPosition">The position of the target during
/// the last time step.</param>
/// <param name="speed">The speed at which the agent can move.</param>
/// <param name="deltaTime">The time elapsed between when the target
/// is in its current position and its previous</param>
/// <returns>The velocity to apply to the agent to perform the pursuit.</returns>
public static Vector2 Pursuit(Vector2 position, Vector2 velocity, Vector2 evader,
Vector2 targetLastPosition, float speed, float deltaTime)
{
    // Get the vector between the agent and the target.
    Vector2 toEvader = evader - position;

    // The time to look ahead is equal to the vector magnitude divided by the sum
    // of the speed of both the agent and the target, with the target's speed
    // calculated by determining how far it has traveled during the elapsed time.
    float lookAheadTime = toEvader.magnitude /
        (speed + Vector2.Distance(evader, targetLastPosition) * deltaTime);

    // Seek the predicted target position based upon adding its position to
    // its velocity multiplied by the look ahead time, with the velocity
    // calculated by subtracting the current and previous positions over
    // the elapsed time.
    return Seek(position, velocity, evader + (evader - targetLastPosition) /
        deltaTime * lookAheadTime, speed);
}
```

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.*

```
/// <summary>
/// Evade - Move from a position factoring in its current speed
/// to predict where it is moving.
/// Based upon the implementation detailed in Programming Game AI by Example page 96.
/// </summary>
/// <param name="position">The position of the agent.</param>
/// <param name="velocity">The current velocity of the agent.</param>
/// <param name="pursuer">The position of the target to evade from.</param>
/// <param name="pursuerLastPosition">The position of the target
```

```
/// during the last time step.</param>
/// <param name="speed">The speed at which the agent can move.</param>
/// <param name="deltaTime">The time elapsed between when the target
/// is in its current position and its previous</param>
/// <returns>The velocity to apply to the agent to perform the evade.</returns>
public static Vector2 Evade(Vector2 position, Vector2 velocity, Vector2 pursuer,
Vector2 pursuerLastPosition, float speed, float deltaTime)
{
    // Get the vector between the agent and the target.
    Vector2 toPursuer = pursuer - position;

    // The time to look ahead is equal to the vector magnitude divided by the sum
    // of the speed of both the agent and the target, with the target's speed
    // calculated by determining how far it has traveled during the elapsed time.
    float lookAheadTime = toPursuer.magnitude /
        (speed + Vector2.Distance(pursuer, pursuerLastPosition) * deltaTime);

    // Flee the predicted target position based upon adding its position to
    // its velocity multiplied by the look ahead time, with the velocity
    // calculated by subtracting the current and previous positions over
    // the elapsed time.
    return Flee(position, velocity, pursuer + (pursuer - pursuerLastPosition) /
        deltaTime * lookAheadTime, speed);
}
```

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.

```
/// <summary>
/// Wander - Randomly adjust forward angle.
/// Based upon the implementation detailed in Dr. Goodwin's
/// "Steering Behaviours" slides on slide 19.
/// </summary>
/// <param name="currentAngle">The current rotation angle of the agent.</param>
/// <param name="maxWanderTurn">The maximum degrees which the rotation
/// can be adjusted by.</param>
/// <returns>The new angle the agent should point towards for its wander.</returns>
public static float Wander(float currentAngle, float maxWanderTurn)
{
    // Since each random call gives a value from [0.0, 1.0],
    // this is a binomial distribution so values closer to zero are more likely.
```

```
    return currentAngle + (Random.value - Random.value) * maxWanderTurn;
}
```

```
/// <summary>
/// Face - Face towards a target position.
/// Custom implementation, not directly based off of any existing code
/// in either Buckland's book or Dr. Goodwin's slides.
/// Given that Vector2 does not have any RotateTowards method,
/// this method is the only to use Vector3 values.
/// </summary>
/// <param name="position">The position of the agent.</param>
/// <param name="forward">The forward vector the agent is visually facing.</param>
/// <param name="target">The position to look towards.</param>
/// <param name="lookSpeed">The maximum degrees the agent can rotate in a second.</param>
/// <param name="deltaTime">The elapsed time.</param>
/// <returns>The quaternion of the updated rotation for the agent visuals.</returns>
public static Quaternion Face(Vector3 position, Vector3 forward, Vector3 target,
float lookSpeed, float deltaTime)
{
    return Quaternion.LookRotation(Vector3.RotateTowards(
        forward, target - position, lookSpeed * deltaTime, 0.0f));
}
```

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			
Green			M	M	M		

	Red	Orange	Yellow	Green	Blue	Purple	Pink
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.

Details

- Created using Unity 2020.3.30f1.
- Created using Easy AI.
- Project setup using the Universal Render Pipeline.
- Project setup using Unity's Input System.