Homework 1

COS498/598 - Video Game AI - Spring 2025

by Zachary Hutchinson

Due Date: Feb 17, 2025, Midnight, Brightspace

Objectives

The objective of this homework is to practice programming AI for a stealth-based game that is both *interesting* and *challenging*. This homework will give you a chance to practice fundamental AI skills such as basic movement, navigation, agent awareness, states, and multi-agent cooperation.

The game consists of three enemy guards (red, yellow and blue) patrolling five important locations. The player's goal is to touch all five locations to win. If any of the three guards touches the player, the AI wins. If the AI moves off the screen, the player wins. If the player moves off the screen, the AI wins.

Task

Your job is program AI for each of the three guards. Your AI **must** have the following properties.

1. Each guard AI must have some unique element or approach. In other words, all three guards patrol the area differently.
2. Although each guard is different, their behavior should contribute to the global strategy of guarding all five locations. You as the designer/programmer should have some guiding strategy when implementing the individual agents.
3. Guards should have at least two states which govern behavior: patrolling and chasing. The patrolling state governs what agents do when they do not see the player. The chasing state governs what they do when they see the player. NOTE: You may have additional states (as many as you like), such as *recently seen* or *reported*, etc.
4. Communication: when a guard sees the player it should use the *comms* dictionary to alert the other guards. In other words, each guard should not be operating solo. There should be some coordination between some of them.
5. You must use text messages to give some sense of personality or narrative to the scene.

Important Advice

Remember: Game AI is not about perfect solutions. You don't want to program an AI that always wins just like you don't want one that cannot win. You don't want to program an AI that is predictable and therefore boring. You want to force the player to think and engage with the AIs contributing to the puzzle. Come up with a narrative and strategy for the situation. Personalize each guard in some way through text and movement such that the player can recognize traits and exploit them. Come up with a short scenario that the player learns about through listening to the guards.

Randomize some elements of the algorithm but keep others fixed.

Create a scene. Create a story. AI is not just about creating a challenge. It is also responsible for a game's personality. Use the text feature to alert the player to a guard's mental or physical state that might help the player decide how to proceed.

By default, the code gives the player a slightly faster speed and turn rate than the guards (which are all the same). You may tweak these values (turn_rate and speed) to achieve the effect you want so long as your tweaks improve the experience of the player. You may also tweak the sight_distance, the initial heading and the starting location (although this last one must be done from the game engine side) of the guards. The player can start at one of eight random locations around the outside of the playing area. Make sure that your starting guard locations do not allow them to see the player from the start.

There are no obstacles in this scenario or objects that block line of sight. Therefore, the scenario is less complex than one found in a typical game of this genre. Your job is to give the player a simple scene to experience and overcome. Force the player to observe patterns or speech before moving forward. It is fine to make some of the goal locations easier than others.

Once the AI discovers the player (if it does), consider how your scene changes. It is not necessary for all guards to bee-line toward the player. Maybe one does. Maybe one flanks. Maybe one moves to block. Don't make them perfect. Allow the player some quirk in guard behavior to break line of sight.

I have not written AI methods for this challenge. I want you to surprise me with an interesting scene and challenge.

The *ai* Method.

Enemy game objects (EnemyBlue, EnemyRed and EnemyYellow) each have an *ai* method. Your AI code should go in the *ai* method. All three derived Enemy classes inherit from the base class *Enemy*. It also has an *ai* method. You may utilize derived class method for unique behavior and the base class method for common behavior. (But you do not have to use the base class method.)

The *ai* method has three parameters: percept, goals and comms.

percept: The percept parameter is a tuple whose length and content varies depending on whether an agent sees the player. If the agent does not see the player, the percept tuple will be: **(False, None)**

If the agent sees the player, the percept tuple will have the following information: **(True, (dX, dY), distance)**, where *(dX,dY)* is a directional unit vector from the agent to the player, and *distance* is the distance from the enemy to the player.

goals: A list with the goal location objects. Take note that this list contains the actual goal objects. This allows the guards to find them using their x,y member variables but also tell whether they have been *touched* by the player.

comms: The comms parameter is a dictionary with three entries, one for each guard. The dictionary is preloaded with three keys: R, B, and Y. Each guard can use this shared dictionary to communicate with the other guards. You can use it in any way you want.

Return value: The *ai* method must return a tuple of three elements. The first two elements of the return are real numbers between -1 and 1. The last value must be either *None* or a tuple of two values. The following describes these values:

0. Index 0 of the return gives the turn value for the guard. The value should be between -1 and 1. A positive value will rotate the guard clockwise. A negative value will rotate the guard counterclockwise. A value of zero will cause the guard to continue facing in the same direction. You may return fractional values between -1 and 1 (e.g., 0.4) if you want the guard to turn at a slower rate.
1. Index 1 of the return tuple gives the move value for the guard. The value should be between -1 and 1. A positive value will cause the guard to move forward along their current heading. A negative value will cause the guard to move backwards. A value of zero will cause the guard to stand still. Like with the turn rate, you can give fractional values for the move rate.
2. Index 2 can contain a message to be printed to the screen. If the guard does

not wish to print a message the third value in the return tuple should be *None*. If the guard wishes to print a message to the screen, it should return a tuple instead of None. The tuple must contain (in this order), the string to print and the time (in milliseconds) to display the message on the screen. The message will be displayed just above the guard, in the guard's color, and it will follow the guard around.

A NOTE ON MESSAGES: Messages are stored in the game engine and repeatedly displayed for the requested duration. You only need to make one request to display a message. Your code should make sure that it does not display more than one message at a time per guard. Message durations are in miliseconds. The *ai* method is called 60 times per second. Your code should keep track of when it last displayed a text message by counting ticks and be carefull only to return a new message once an old one has expired. I suggest using a countdown timer for each guard. I also suggest that you keep messages short.

Here are some return value examples:

(0.0, 1.0, None): would cause the guard to move forward without turning. No message would be displayed.

(0.0, -0.1, ("I didn't sign up for this!", 3000)): would cause the guard to move backward at 1/10th its speed and display a message for 3 seconds.

Deliverables

You need to submit two things.

Code:

Submit to Brightspace both your hw1_main.py and gobjs.py files containing your ai routines.

Write-up:

A detailed write-up (approximately 1-2 pages) in **PDF** format describing a) your experience working on this homework and b) your AI. I will discuss my expectations for a write-up in class. I suggest you take notes as you work.

Grading

Your code **must** work. Code that crashes will get a zero. Therefore, test your code!

There are five bullet points under the Tasks section above describing what your code must do. Each aspect is worth 2 points. Your write-up should highlight what you did to satisfy each of those five aspects and your code should mirror what you describe. If they aren't in your write-up, we assume you didn't do them. Obviously, if they aren't in your code, you didn't do them.

A last-minute, after-the-fact, sloppy write-up will lose points.

Thou Shalt/Shalt Not

In general, your AI routines should depend only on the information already provided. You should not need to pass in more information via the class **init** methods or add additional parameters to the agent's *ai* methods. But this is a loose restriction. If you really want to make changes to the engine, run them by me.

You may change the enemy classes (base and derived) to store extra info. You may add more methods to them.

Again, you may rearrange the locations of the various objects (even the goal locations) to suit the narrative you want to portray.

The player starts in one of eight random locations. You may change the player start locations. You may even fix it so that the player always starts in the same location. Your guard patrol code should not use this information. It should assume the player starts anywhere.

You are not allowed to teleport the guards or instantly change their facing. You must use the return value of the *ai* routine to turn and/or move the guards. In other words, don't change the x, y or heading of the guards from within the *ai* routine. Allow the engine to do this.

Your code should not require any additional libraries other than pygame and the standard modules that come with a python3 installation.