# CS 4150/5150: Game Artificial Intelligence

# Project #1

# Perception

# Due: 6:00pm, Tuesday, 2/21/23

## Overview:

Build a new “Rogue” character for *Crash Loyal*, our home-brew clone of the popular mobile game *Clash Royale*. This character will hide behind Towers and Giants, and then spring out to attack its enemies, doing massive damage with its first strike. When no enemies are around, it will go back into hiding, waiting for new opponents to arrive.

## Starting Code:

The updated source code for this project can be found in Project2.zip. This version of the game adds the Rogue as a new type of mob. You can search the code for “Project 2” to get some hints of places to start making your own changes.

## Desired Behavior:

The Rogue is a sneaky character who hides out behind Giants, and then springs out to protect them while the Giant continues on to take out enemy Towers. Note that because of its small size, it can’t attack enemy Towers, only mobs (this restriction should already be implemented, but you’ll want to test and make sure it’s working for you!).

Its behavior should work as follows:

* If there is a friendly Giant or Tower blocking every enemy unit (including both mobs and buildings) from seeing the Rogue, and this has been the case for at least 2 seconds, then the Rogue becomes “hidden.”
  + Only living Giants and Towers can hide the Rogue – dead ones cannot.
  + Only living enemies can spot the Rogue.
  + Only enemies that are within their sight radius (a new constant in EntityStats.h) can see spot the Rogue.
  + SUGGESTION: I didn’t add a constant anywhere for that 2 second magic number, but you probably should add it (and any other constants that I forgot to add) to EntityStats.h, so that it can be tuned like everything else.
* When the Rogue is hidden:
  + If there is an enemy unit within spring attack range, then it will spring out, moving extra fast and doing increased damage on its first attack. See EntityStats.h for these (and other) new constants.
  + Otherwise, if it is currently hiding behind a Giant (see below for how this condition gets set), then it will move along with the Giant, staying within its hiding distance (see EntityStats.h), and moving around the Giant to try to stay hidden from all enemy units that are within its sight radius but not within spring attack range (sight radius is another new constant in EntityStats.h).
    - NOTE: You may have to adjust the hiding distance to get this to work right. It should be close enough to the Giant that, when the Giant is attacking a Princess Tower, the Rogue can hide from both that Princess Tower and the King Tower. It should be hidden from the other Princess Tower because of the Princess Tower’s sight radius (but only barely). If you need to tune the constants a bit to make that happen, that’s fine… just be sure to document what you did in your written documentation.
  + Otherwise, if it is currently hiding behind a friendly Tower (see below for how this condition gets set), then…
    - If there is a friendly, living Giant within “prefer Giant range” (see EntityStats.h) then it should switch to hiding behind the Giant. Thus, when you drop a Giant close enough to one of your Towers, any Rogues hiding there will switch to hiding behind the Giant – but the Giant has to be quite close for this to happen, otherwise they stay at the Tower.
    - Otherwise, it should continue to hide behind the Tower, staying at the hide distance and moving around the Tower as needed to hide from enemy units that are not within spring attack range, just like it does when hiding behind a Giant.
  + Otherwise, if there is a friendly, living Giant within its sight radius, it will move to the Giant, trying to hide behind it as it moves. Note that it only does this when not already hiding behind a Tower – otherwise, the Giant would need to be withing “prefer Giant range.” Once it reaches that Giant it is considered to be “hiding behind the Giant,” even if it is not yet actually hidden (i.e. has not remained occluded or out of sight range of all enemies for 2 seconds).
  + Otherwise it should move to the closest friendly Tower. Once it reaches it, it is considered to be hiding behind that Tower (see above).
* When the Rogue is not hidden:
  + If it is currently attack an enemy, it will continue to attack that enemy until one of the other is dead.
  + Otherwise, if there is an enemy mob within its sight radius, it will move to that enemy and attack it (just like the swordsman does, for example).
  + Otherwise, if there is a living, friendly Giant within its sight radius, it will move to the Giant and attempt to hide behind it.
  + Otherwise, it will move to the closest friendly living Tower and attempt to hide there.

## Grading Guidelines:

The assignment will be graded based on how well it works, not on how it is implemented. Grading will be done as follows:

* (30 points) Documentation
* (25 points) The rest of the game continues to work as expected
* (30 points) The Rogue functions as described above. Partial credit is available – get it to work as well as you can! I strongly recommend implementing this a piece at a time, checkpointing each time you get a feature working, so that you have something to turn in even if not everything works.
* (25 points) Aesthetics – does the behavior that you built look and feel “right.”

Note that there are 110 total possible points – extra credit is possible.

Also, note that the collision avoidance code you wrote for Project 1 is optional. If you want to use it, you’ll need to port it over to the new project. This isn’t required, but may be helpful (especially if you implemented steering forces for Project 1) to help you build the behaviors where the Rogue tries to stay close to the thing its hiding behind, move with it, and also move around it to stay hidden, all at the same time.

## Deliverables:

Your deliverables should be zipped up and submitted via Canvas. They should include:

1. All source code needed to compile and run the game.
2. Written documentation that describes the approach you took. This should specifically include:
   1. How your solution works.
   2. Anything you did beyond what is described above (e.g. bringing in your collision code from project 1, tweaking constants, adjusting behavior for a better look and feel, etc.)
   3. What parts you think work particularly well.
   4. What things you wish you had time to improve.

In addition, the TAs will schedule a meeting with each of you, during which they’ll play-test your project while discussing your approach with you. During this time you’ll be able to discuss the feel that you were going for as you interpreted the guidelines given above.