Dept. of Computer Science & Software Eng., Concordia University

COMP 476/6331 --- Winter 2025

Advanced Game Development

Quiz Assignment --- Due Sunday, March 9, 2025, @23:59

Title: Rock, Paper, Scissors, Lizard, Spock: Battle Arena

In this game, you are going to create an extension of the rock-paper-scissors game called “Rock, Paper, Scissors, Lizard, Spock”.

https://bigbangtheory.fandom.com/wiki/Rock,_Paper,_Scissors,_Lizard,_Spock

In this game, there will be 10 rocks, 10 papers, 10 scissors, 10 lizards, and 10 Spocks on a platform, i.e., a gameplay arena, with 10 randomly distributed obstacles. The game has simple rules:

- Rock beats scissors (rock crushes scissors)
- Rock beats lizard (rock crushes lizard)
- Scissors beats paper (scissors cuts paper)
- Scissors beats lizard (Scissors decapitates Lizard)
- Paper beats rock (paper covers rock)
- Paper beats Spock (Paper disproves Spock)
- Lizard beats Spock (Lizard poisons Spock)
- Lizard beats paper (Lizard eats Paper)
- Spock beats Scissors (Spock smashes Scissors)
- Spock beats rock (Spock vaporizes Rock)

To beat a unit, one game object has to collide with another game object tagged as an enemy (see below for terminology). For example, a rock game object beats the scissors - if these two objects collide.

The aim is to eliminate one of the factions (rocks, papers, scissors, lizards, and Spocks) and not to lose. To eliminate the faction, all the units of the faction must be beaten.

Rule to lose: If one of the factions is completely eliminated (beaten), the game ends. For instance, rocks will try to seek all the scissors and lizards while fleeing from papers and Spocks. If there are no scissors left in the arena, then the scissors lose the game. Others win the game.

Terminology: There are four important terminologies that must be also used in the code:

Factions: A group of game objects shares the same tag. There are five factions in this game: ‘Rocks’, ‘Papers’, ‘Scissors’, ‘Lizards’, and ‘Spocks’. For instance, all the game objects, i.e., units, under the tag of ‘rock’ are grouped as ‘Rocks’.

Friendly Units: These are the individual units with the same tag. For a rock game object, all the other game objects tagged as rock are friendly units.

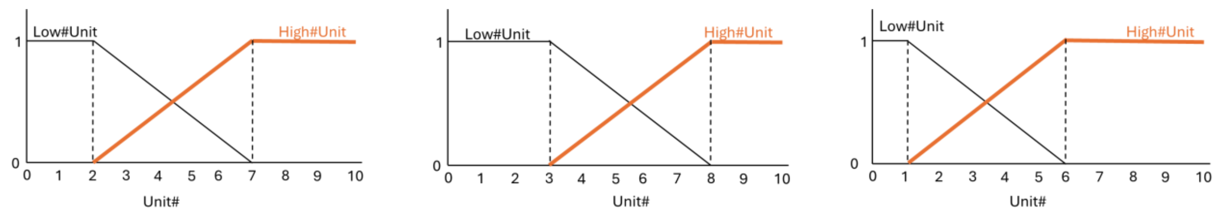
Enemy Units: These are the units that the game objects try to flee. If an enemy game object collides with an enemy game object, it is eliminated. For a rock object, all the other game objects tagged as paper and Spock are enemy units because paper beats scissors and Spock vaporizes rock.

Target Units: These are the units that the game objects try to seek and beat. If the enemy game object collides with the game object, the target game object is eliminated, e.g., destroyed. For a rock object, all the other game objects tagged as scissors and lizards are target units because rock beats scissors and rock crushes lizard.

Coding: You are going to write Group AI (`GroupAI.cs`) and Individual AI (`IndividualAI.cs`) code for units, as well as write other small pieces of code to run the game smoothly. In GroupAI, you will decide the faction's speed (called `aggressiveness`) to set the speed units. In Individual AI, each unit's speed will be changed if they are chased by an enemy unit.

Group AI: Each faction (rocks, papers, scissors, lizards, spocks) must have a game object called `GroupAI.cs.` Here, you are going to implement a Fuzzy Logic algorithm that calculates the aggressiveness (the speed) of your units.

Fuzzification: Use Figure 1 to find the degree of the membership function based on the number of units. Be careful; each membership function is different.



(a) **(b)** **(c)**

Figure 1- Membership functions for (a) friendly units, (b) target units, and (c) enemy units.

Rule Evaluation: Implement the following rules:

If (High#EnemyUnits AND High#FriendlyUnits) then average speed

If (High#EnemyUnits AND Low#FriendlyUnits) then aggressive

If (Low#EnemyUnits OR High#TargetUnits) then average speed

If (Low#EnemyUnits OR Low#TargetUnits) then move calmly

If (High#TargetUnits OR High#FriendlyUnits) AND NOT High#EnemyUnits then aggressive

If (Low#TargetUnits OR High#FriendlyUnits) AND NOT High#EnemyUnits then average speed

If (Low#TargetUnits OR Low#FriendlyUnits) OR Low#EnemyUnits then move calmly

If needed, you can add more rules to fuzzy logic. If necessary, you can change the output of the rule, e.g., from aggressive to moving calmly.

Defuzzification: Use normalized *blending* based on membership to calculate the crisp output, i.e., aggressiveness. Use the following speeds:

> aggressive = 10 m/update
> average speed = 6 m/update
> move calmly = 2 m/update

The speed of a unit can be maximum 10m/update without boost (see below for boost speed).

IndividualAI (30%): Each unit must have an `IndividualAI.cs` that decides what the unit does. For this, you need to write and implement the following:

+ If there are no enemies around the unit within 5m, the speed of the unit will be set to `aggressiveness`, which GroupAI assigns.

+ The unit will find the closest target.

+ The unit will chase the closest target to catch it.

+ If there is an enemy in close proximity (less than 5 m), the speed of the unit will be set to the current `aggressiveness` speed +2 for 2 seconds. This is called the `boost` speed.

+ The unit will start fleeing.

+ When 2 seconds ends, the speed of the unit returns to its `aggressiveness` speed. The unit will continue fleeing if the enemy is still less than 7.5 m away.

+ The unit can get this speed increase (`boost`) again after 2 seconds.

+ If the enemy is more than 7.5 m away, the unit stops fleeing and returns to chasing the closest enemy.

+ Similarly, the enemy stops chasing the unit if the distance between the enemy unit is more than 7.5 m. In this case, the enemy must find the other closest target.

+ If you want, you can also add strategies or tactics to the individual AI to create a more exciting game.

+ The unit will avoid colliding with obstacles (see below).

Game design (5%): Aside from the aesthetics, you need to implement the following two rules:

1) The units must stay within a 75 m radius.

In other words, you need to select a center, e.g., (0,0,0), and all the units can at most reach 75 m away from the center. The current gameplay arena is 100m x 100m, so here are the several options you can consider:

- You can turn the floor into a bowl shape so that units cannot reach the edges.
- You can create a fire, toxic fog or something similar and reduce the radius of the playable space every second, as in Fortnite or PUBG.
- You can add a wall and run a collision avoidance algorithm.

- You can teleport the objects back to the center or a random place in the game.
- You can create a void after 75m and let game objects fall into that void.
- Any other creative design that comes to your mind.

Since the game must follow logic and the behavior of the units must be realistic, you need to consider your design carefully. While basic and simple methods might work, students with complex, marginal, or fancy designs will receive higher grades.

2) The units need to avoid hitting obstacles.

You must add (at least)10 obstacles with different shapes and sizes within the gameplay arena. The units must avoid hitting these obstacles. If they hit these obstacles, they are eliminated. Please see the tip regarding obstacles below.

Other Rules:

- You can add new classes, methods, events, etc., if they help you to improve your game.
- The game was developed on an Intel Core I5-10300H CPU @2.5GHZ 8 GB RAM RTX 3060 (Nitro 5) Computer. When you develop your game, you might need to optimize your code or AI behaviors accordingly.
- At the beginning of the game, the player selects their faction and places each unit in the gameplay arena.
- Do not forget to add basic UI elements, such as the number of units, ‘pause game’, ‘quit game’, or ‘start new game’.
- You are going to design how the units are placed in the game, e.g., you can put them close to each other, randomly generate new positions for each game, etc., to create more exciting gameplay.

Grading:

- Fuzzy Logic for group AI:	30%
- Decision tree of Individual AI and movement AI:	30%
- Richness of AI behaviors:	10%
- Game design (keeping objects in 75m radius and obstacles)	5%
- General aesthetics and relative performance:	5%
- QnA with the Grader:	20%

You are going to submit both .exe file and Unity files of the game. Only working .exe games will be graded. The marker must be able to run your program if it works to evaluate it, so you must give any instructions necessary to get your program running. If your program does not run, we will not debug it.

Note on marking: As designed, there can be a wide variation in fulfilling the requirements of this assignment; just barely meeting the requirements will not result in a full mark. For example, if your program only barely has the features listed in AI, do not expect full marks. Behaviors, where there is an obvious extra effort made to make the interaction between NPCs more realistic/interesting/appealing, may expect full or nearly full marks (this is the purpose of the 10%

for "richness of AI behaviors", and the 5% for "relative performance").

Tip: If you want to get a higher grade, you can consider the 'obstacles' not only simple 3D shapes, such as boxes or spheres, but more like small mazes, where you can include tactical AI. This would increase your score regarding the richness of AI behaviors and relative performance.

Note: Students are expected to adhere to ethical standards, and any form of plagiarism will result in severe consequences. Students who cannot answer questions in Q&A might not get any grades.

Submission Deliverables (Only Electronic submissions accepted):

1. Submit your whole Unity project, including data files, if any, along with auxiliary files needed to quickly get your program running, and any other instructions for compiling/ building/running your program. If the file size is too large, you can delete, e.g., unnecessary files, GitHub links, and unused models. Also, check the settings of your application, and be sure that you set the comparison to maximum. You can also try splitting files. The source code (and brief readme, explained below) must be submitted on Moodle in a **zip format** with all the required files (ex.: **YourIDnumber.zip**). You can also divide .zip file to smaller portions. ***Do not e-mail the submission or send a link.***

2. You are also going to upload an .exe or similar version of your working game. You can create an executable file (.exe file for PC), zip it, and upload it to Moodle. To be on the safe side, you can upload both .exe and WebGL games to Moodle and, moreover, upload your game online, where we can track your last update. This working version of your game will be evaluated before demonstrations.

   DO NOT ONLY SUBMIT A .exe FILE.

3. Demonstrate your **Unity project** and working program to the TA and marker in your lab sessions. It is mandatory for all students.

4. Included in your zip file will be a readme about your program explaining the characteristics of the behaviors you have defined as a PDF file and any special features that you would like us to consider during the evaluation.

Good luck!