Project 3 Report (By Giselle Kian & Ali Rajabi)

PART 1 (Inverse Kinematics Scenario)

In **video #1**, the first feature which is "**Single-arm IK**" has been implemented. The algorithm used for the implementation of this part is simply inverse kinematics (IK) to update the angle difference derived from both dot product and cross product, and forward kinematics to update the position of the joints start and end points.

In **video #2**, the second feature which is "**Multi-arm IK**" and "**Joint-Limits**" has been implemented. The point of this part is two things; first, angle limits have been added to the structure and joints cannot rotate and bend more than an angle cap and less than an angle minimum. The second point is that different joint speeds have been added to the structure so that every single joint moves and rotates with a slower speed.

In **video #3**, the third feature which is "**IK Obstacles**" has been implemented. In this video, there are two different objects, one is a circle and another one is a box. It can be easily seen that the single arm can interact and collide with both obstacles separately.

In **video #4**, the fourth feature which is "**User Interaction**" has been implemented. By pressing the "b" button, the box can easily move, or by pressing the "c" button the circle can easily move within the canvas. Then, it can be observed that the single-arm can interact with both obstacles again and also collide with them.

In **video #5**, there are four features that have been implemented, which are "**3D Simulation**", "**3D Rendering & Camera**", "**Skinned Model**", "**IK + Character Animation**".

In **video #6**, for the "**Full Crowd Simulation"** simulation challenge, the Boid algorithm has been utilized. This Boid script provides a comprehensive set of functionalities for simulating flocking behavior in a Unity environment. It encompasses features such as target following, flocking alignment, cohesion, separation, and collision avoidance. Understanding and customizing these behaviors can be crucial for creating dynamic and lifelike flocking simulations in Unity.

==========

List of the tools/library you used:

Processing:
I used "Processing" and the "Vec2" library that was created before, with adding some functions like dot product, cross product and so on.

Unity 3D:
For a 3D character simulation in Unity, I used the Unity Editor library for writing the C# scripts.

==========

Brief write-up explaining difficulties you encountered:

1. The first difficulty that I faced was that the single-arm got stuck to the obstacle once there was a collision between any joint of the arm and the obstacle. The reason was defining the variables “old_a2”, “old_a1”, and “old_a0” as global variables while they should have been defined as local variables in the “solve” function.

2. The second difficulty that I encountered was with multi-arm IK, where I could not get two arms to work well separately, while they got stuck together sometimes. However, both arms were like following the goal which was the mouse position in the code.

3. For a 3D character simulation in Unity, I struggled with how to implement my own IK solver on the character bones, and to make it walk on its own using the controllers, since it may get confusing on which game objects the scripts need to get attached to.

4. For the Full crowd simulation challenge, Implementing effective collision avoidance that considers dynamic obstacles and the environment has been very complex. Ensuring that boids navigate around obstacles without getting stuck or exhibiting erratic behavior required careful algorithm design.