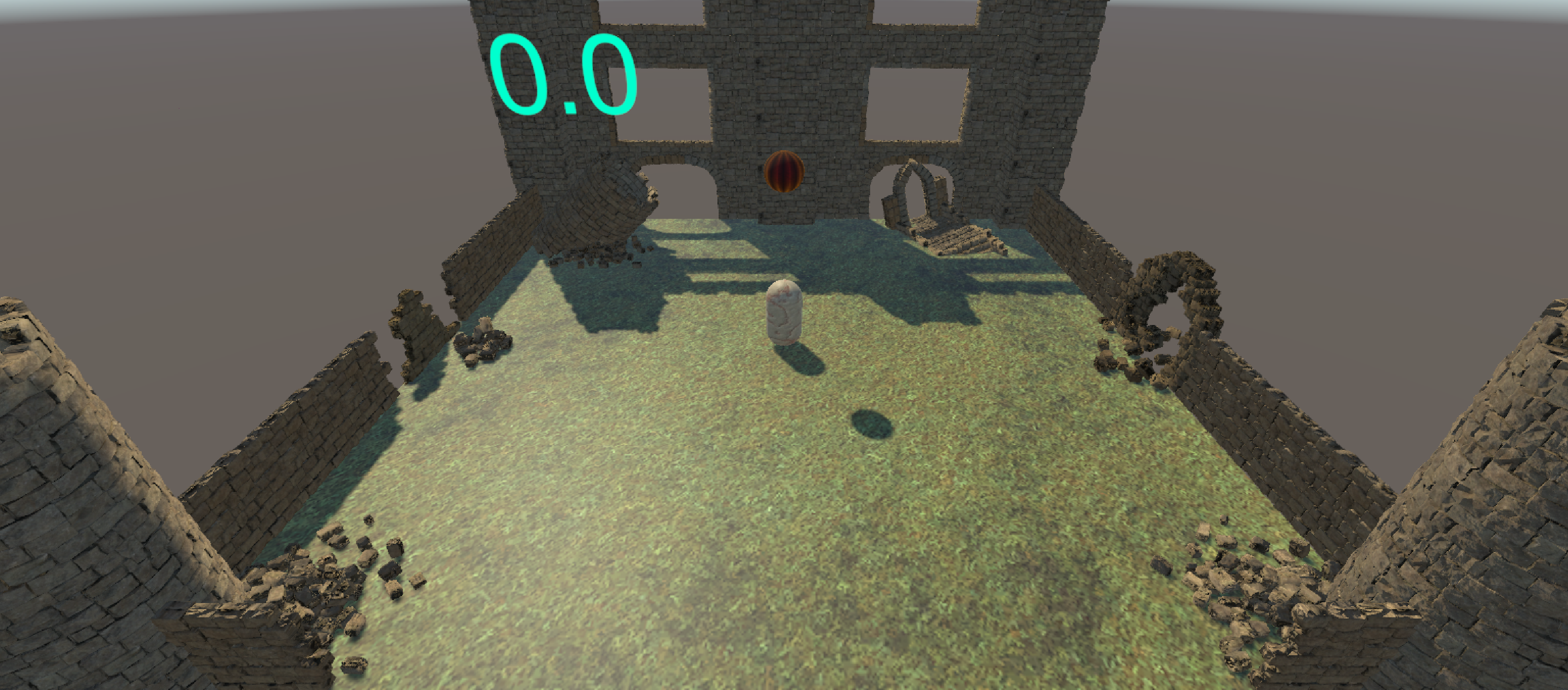
**Game AI created based on Unity reinforcement learning tool MLAgents**

**Tools: Anaconda + ml-agents-release\_18 + pytorch**

**Game:** We set up a scene where a ball randomly appears above a player. The player can move in three directions: forward, backward, and sideways. Each time the player successfully juggles the ball, they earn 0.1 points. The game ends if the ball hits the ground or if the player moves out of bounds.

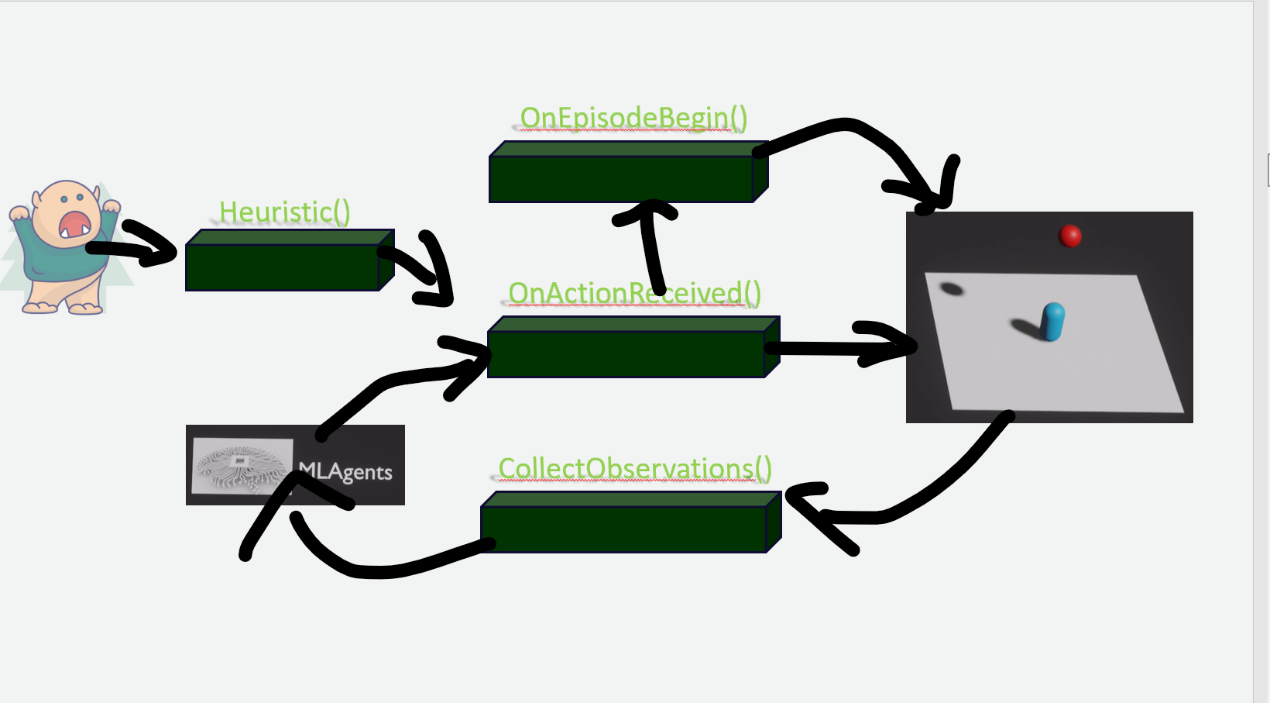


ML-Agent Installation and Training process:

<https://blog.csdn.net/hnbxs/article/details/136110026>

How to use: First configure the environment of MLAgents, then import the code into Unity, and start running after entering the Final.unity file in FinalAsset. Detailed Code Analysis Report. (The new version of MLAgents requires the Barracuda package to be pre-installed in Unity.) The trained NN Model in this project is stored in the Models folder.

**MLAgents:**



First, Collectobservations() will collect from the game environment, such as the position and velocity of the ball, and pass this information to ML-Agents.

Then, ML-Agents will use the submitted environment information to calculate an action array.

This action array determines how the player can move in the three dimensions in our scenario.

In the second step, the action array returned by ML-Agents will contain three elements representing the movement information for each direction.

How the player moves specifically depends on the programming implementation. The entire second step is managed by ML-Agents as long as we configure the inputs and outputs correctly.

In the third step, the action array from the previous step is submitted to the OnActionReceived method. This method executes the game's operations, such as moving the player, checking if the ball is juggled, and determining if the game has ended. If the game ends, it calls OnEpisodeBegin to reset the game environment. If the game doesn't end, it loops back to the first step and continues.

While writing the program, we need to test the game environment. This is where the Heuristic method comes in. It converts our manually inputted commands into the array format that OnActionReceived can process. During this time, the Initialize method is also called to initialize the agent's operations, such as setting initial values.

Detailed Code Analysis Report



* Various Unity and ML-Agents namespaces are imported.
* The JuggleAgent class inherits from Agent, which is part of the ML-Agents toolkit.



* ball: Reference to the ball's Rigidbody component.
* player: Reference to the player's Rigidbody component.
* speed: A control signal multiplier.
* display: UI element to display the cumulative reward.
* diff, previousDiff, previousY, collied: Variables to track the ball-player interaction and detect successful juggles.
* Note: diff = ball.transform.localPosition.y - previousY; x is the horizontal coordinate, z is the vertical coordinate, and y is the variable for the direction of the JUMP.



* Detects when the ball collides with the player and sets collied to true.



* Initializes the player's Rigidbody component.



* Collects various observations about the ball and player, including position, velocity, rotation, and angular velocity.



First of all, the ball will fall down, the y-axis keeps decreasing, and the change is negative, the successful bouncing of the ball in the process of bouncing up, the change of the y-axis is positive. But there is an exception, it may be the top of a player with a ball back and forth up and down movement, the score has been getting bigger, in order to solve this problem we add a collision detection. So we need to define four class variables, the first one is diff which records the change of player's Y-axis, the second one is previousDiff which keeps the previous diff, and then define previousY which records the player's previous Y value, and define a bool collide which indicates whether a collision has occurred, and the current player's coordinate Y is subtracted from the previous Y. If a collision occurs, the current player's Y coordinate is subtracted from the previous Y coordinate, and the previous Y is subtracted from the previous Y coordinate. If diff is greater than 0, and previousDiff is less than 0, and collide is true, then the collision will be counted as a collision.

* Receives an action buffer and converts it into a control signal for player movement.
* Adds force to the player based on this control signal.
* Calculates the difference in the ball's Y-position to detect successful juggles.
* Rewards the agent for successful juggles.
* Ends the episode if the ball hits the ground or the player moves out of bounds.
* Updates the display with the cumulative reward.



Ball Initialization

* ball.transform.localPosition: Sets the ball's position to a random location within a 10m x 10m square above the player (y = 5.0f).
* ball.velocity: Sets the ball's initial velocity to zero.
* ball.rotation: Resets the ball's rotation to the default orientation.
* ball.angularVelocity: Sets the ball's initial angular velocity to zero.
* Player Initialization
* player.transform.localPosition: Positions the player at the origin (y = 1.0f, which is Vector3.up).
* player.velocity: Sets the player's initial velocity to zero.
* player.rotation: Resets the player's rotation to the default orientation.
* player.angularVelocity: Sets the player's initial angular velocity to zero.
* Variable Reset
* diff, previousDiff, previousY, collied: Resets these variables to their initial values to prepare for a new episode.



Heuristic Method

* Provides manual control inputs for testing the agent without using the ML model.
* Maps the horizontal, vertical, and jump inputs from the user to the action buffer.
* Input.GetAxis("Horizontal"): Maps to left/right movement.
* Input.GetAxis("Vertical"): Maps to forward/backward movement.
* Input.GetAxis("Jump"): Maps to upward movement (jumping).